Table of Contents

Day 1 Multi-Threading, Advanced UI/ UX, Android Studio, Gradle & Dependencies,

Day 2 Networking, Content Provider & REALM database, MEAN Stack and Android

Day 3 Testing, TDD, Automated Testing Tools for Android,

Day 4 Architecture and Design, Design Patterns, Agile Development, Tools for Design

Day 5 Cloud, Big Data and Android, RxAndroid, Clean Architecture,

https://www.raywenderlich.com/

Topics

- **Day 1** **Best Practice in Coding, Clean Coding Style + Android**

The Most Important Skill Any Software Developer Can Have Is… Knowing How to LEARN
The Faster I Learned, The More I Retained

10 Steps To Learn Anything Quickly

Discovering these techniques for yourself will change your entire outlook on software development—just like it did for me.
Instead of feeling overwhelmed by the flood of "must-know" new technologies, frameworks and tools that you're bombarded with every day, you'll adopt a Zen-like attitude toward the churn.
You'll relax in the knowledge that you DON'T have to jump on every new trend (or risk getting left behind).
You'll have the confidence that you can pick up a new technology and know it back to front in a few days or a couple of weeks.
Instead of feeling scattered and distracted, your learning time will have laser-like focus and structure.
And you'll finally experience the satisfaction of actually FINISHING. You'll know when you've learned enough and you're ready to move on.

This 10-step system will show you:

- How breaking out of the "learning cattle chute" can help you master even the most challenging technologies quickly.
- The reason why including a mandatory "play time" in your study sessions can rocket you up the learning curve faster than you thought possible—even though you feel like you're just goofing off.
- A reliable heuristic for knowing what to read—and what to ignore. That's right: Dumping that massive stack of unread books and deleting your Insta-paper queue will actually make you a faster learner. (This will free you from "Amazon reading list guilt" forever.)
- How to harness your natural creativity and curiosity for better, faster learning (with zero frustration and overwhelm).
- How to chart your own course for learning a new topic. This will free you from the soft tyranny of book authors and trainers who think they know the best way for you to learn.
- The 3 critical questions you must answer when you start learning anything new. (Don't waste weeks wandering around lost and confused. These questions will point you in the right direction, every time.)
- How to quickly get a "toehold" on any new topic before you dive deep. This technique lets you skip right past the part where your head spins from overwhelm.
- How to break out of the "infinite reading loop," where the more you read, the more your reading list grows.
- The way to set yourself up for learning success. When you do this, you'll know when you're done with a topic and ready to move on to new challenges.
- How you can use your new learning superpowers to quickly gain respect as a leader and an authority. Your team will look to you when it's time to evaluate new technologies, and you'll have top-notch employers beating down your door.
- How to learn more by learning less. (This is critical to eliminating overwhelm—and actually putting what you're learning into practice.)

- **Session 1 Part 1** **Android Studio 2.3.3 + Gradle 4.1**
  - Objective – Staying Sharp - https://simpleprogrammer.com/products/learn-anything/
  -
  - Getting Started with Android Studio
    - http://www.vogella.com/tutorials/Android/article.html

- http://www.vogella.com/tutorials/Android/article.html#android-studio-overview-and-installation
- https://www.tutorialspoint.com/android/android_studio.htm
-
- https://developer.android.com/studio/known-issues.html Known Issues
- http://tools.android.com/knownissues
-
- https://developer.android.com/studio/releases/index.html
-
- https://stackoverflow.com/questions/tagged/android-studio-2.3
- https://simpleprogrammer.com/2011/01/08/solving-problems-breaking-it-down/
- https://www.hackerrank.com/programming-interview-questions
-
- https://simpleprogrammer.com/products/learn-anything/
-

- Tips and Tricks of Using Android Studio
  - Keyboard Shortcuts - https://developer.android.com/studio/intro/keyboard-shortcuts.html
  - https://github.com/nisrulz/android-tips-tricks Cheat Sheet
  - https://www.tutorialspoint.com/android/android_studio.htm
  -
  - https://medium.com/@mmbialas/50-android-studio-tips-tricks-resources-you-should-be-familiar-with-as-an-android-developer-af86e7cf56d2
  - https://www.youtube.com/watch?v=XCo-xWyqLQo
  - https://github.com/nisrulz/android-tips-tricks
  - https://medium.com/@mmbialas/50-android-studio-tips-tricks-resources-you-should-be-familiar-with-as-an-android-developer-af86e7cf56d2
  - https://stackoverflow.com/documentation/android-studio/2228/android-studio-tips-and-tricks#t=20170924035653643 7156
  -
  -
- Android Studio 2.3.3 + Gradle 4.1 Tutorial
  - https://www.javaworld.com/article/3104595/android/android-studio-for-beginners-part-4-advanced-tools-and-plugins.html
  - https://www.udemy.com/sisoft-android-app-development-advance/
  - https://www.youtube.com/watch?v=UqtsyhASW74
  -

All libraries in Android

- https://android-arsenal.com/
- https://www.pivotaltracker.com/ keep track of your android project
-

o Session 1 Part 1 Android Studio 2.3.3 + Gradle 4.1

- Interview Questions in Android
  - http://javarevisited.blogspot.in/2015/01/top-20-string-coding-interview-question-programming-interview.html
  - https://www.glassdoor.co.in/Interview/Facebook-Android-Developer-Interview-Questions-EI_IE40772.0,8_KO9,26.htm?countryRedirect=true
  -
  - https://www.careercup.com/page?pid=problem-solving-interview-questions
  - https://www.hackerearth.com/challenge/hiring/android-hiring-challenge/

- https://blog.aritraroy.in/what-my-2-years-of-android-development-have-taught-me-the-hard-way-52b495ba5c51
-
- https://www.testdome.com/tests/android-online-test/49
- https://github.com/MaximAbramchuck/awesome-interview-questions
- https://www.testdome.com/tests/java-android-online-test/51
- https://www.toptal.com/android/interview-questions
-

o **Session 1 Part 2 Connect Bitbucket, Jira to Android Studio**

- http://www.vogella.com/tutorials/AndroidBuild/article.html
- http://tools.android.com/tech-docs/new-build-system/user-guide
- https://rominirani.com/gradle-tutorial-part-6-android-studio-gradle-c828c5639bb
- http://techdocs.zebra.com/emdk-for-android/6-3/tutorial/tutCreateProjectAndroidStudio/
- https://stackoverflow.com/questions/30817871/android-studio-is-slow-how-to-speed-up
- http://www.viralandroid.com/2015/08/how-to-make-android-studio-fast.html

General Info

- https://en.wikipedia.org/wiki/List_of_mobile_phone_makers_by_country
- https://www.androidcentral.com/aosp
- https://source.android.com/

Tips and Techniques in Advanced Coding in Android

- http://www.androidauthority.com/tips-tricks-new-android-app-developers-336322/
- https://www.toptal.com/android/top-10-most-common-android-development-mistakes
- https://dzone.com/articles/few-tips-beginning-android
- https://dzone.com/articles/10-attractive-android
- http://blog.edx.org/15-tips-tricks-android-app-developers-2017
- https://medium.com/@mmbialas/50-android-studio-tips-tricks-resources-you-should-be-familiar-with-as-an-android-developer-af86e7cf56d2
- https://www.codementor.io/codementorteam/5-ways-to-make-learning-android-development-easier-aak4812o3
- https://androidexample.com/?gclid=CjwKCAjwjJjOBRBVEiwAfvnvBLmcOQRsHCQhDC_KNd4okYG1USr_RJq35wLpeHu8DODjtzRBwE2RzxoCVc8QAvD_BwE

Day 1 Best Practice in Coding, Clean Coding Style + Android

Session 2 Building Better Code Blocks + Android

- https://developer.android.com/training/advanced.html
- https://in.udacity.com/course/advanced-android-app-development--ud855
- https://www.udemy.com/advance-android-programming-by-9i-technologies/
-

Some Notes for Trainers

- https://blog.aritraroy.in/20-awesome-open-source-android-apps-to-boost-your-development-skills-b62832cf0fa4
- https://bugfender.com/blog/the-best-android-app-development-learning-resources/
- http://androiddeveloper.galileo.edu/2017/02/08/the-top-10-books-android-programming-2017/
- https://www.quora.com/What-are-good-sources-for-learning-advanced-Android-development

  - Architecting Android…The clean way?
  - android10/Android-CleanArchitecture

- Building Android Apps — 30 things that experience made me learn the hard way
- https://www.reddit.com/r/android...
- konmik/konmik.github.io
- ppicas/android-clean-architecture-mvp
- Tasting Dagger 2 on Android
- Retrofit
- Butter Knife
- orfjackal/retrolambda
- Espresso
- Android Studio 2.2

- https://www.quora.com/How-do-I-learn-advanced-android-programming

- Follow android blogs and go for Graphics, gamming, OpenCV, OpenGL, custom views and Design Pattern.
  stylingandroid.com
  Cyril Mottier
  Chris Banes
  Jake Wharton
  Sriram Ramani
  Android UI Patterns
  Android development
  Grokking Android
  Android Developers Blog
  The Radioactive Yak
  GeekYouUp's Mobile Blog

Day 1 Best Practice in Coding, Clean Coding Style + Android

Session 2 Building Better Code Blocks + Android

- https://www.edx.org/course/professional-android-app-development-galileox-caad003x

Learn mobile application development on the Android platform by acquiring strong knowledge of Android SDK and different versions of Android. You will gain familiarity with RESTful APIs to connect Android applications to back-end services and will practice with Geny motion emulator.
As a part of this course, you will create widgets, customize list views and create 5 applications using Facebook, Twitter, maps & location based services. You will test Android-based **mobile applications** using Android testing tools such as Mockito.
This Android course is taught by a group of **Google Developer Experts** and other industry professionals, who develop innovative mobile apps.
This course is part of the GalileoX Android Developer Micro-Masters Program that is specifically designed to teach the critical skills needed to be successful in this exciting field and to prepare you to take the Google Associate Android Developer Certification exam. In order to qualify for the Micro-Masters Credential you will need to earn a Verified Certificate in each of the four courses as well as Final Project.
See more about Professional Android App Development

What you'll learn

- Firebase and Android

- Model-View-Presenter (MVP)
- Clean Architecture Android
- Create and display a notification to the user
- Building at least 5 android applications
  - #01 App: Building a Basic Chat Application
  - #02 App: Create a Twitter App
  - #03 App: Integrating Facebook with my App: Facebook Recipes
  - #04 App: Your social network of photographs!
  - #05 App: Building a note-taking app for android
- How to assure the best possible performance, quality, and responsiveness of the application
- Integrate code from an external support library
- Use the system log to output debug information

View Course Syllabus

- https://www.safaribooksonline.com/library/view/advanced-androidtm-application/9780133892420/

Broadcast Receiver

- http://skillgun.com/question/543/android/receivers/what-is-the-difference-between-sendbroadcast-sendorderedbroadcast-sendstickybroadcast
- https://stackoverflow.com/questions/3156389/android-remoteexceptions-and-services
- https://developer.android.com/guide/practices/index.html
- https://developer.android.com/training/best-performance.html
- https://blog.mindorks.com/android-development-best-practices-83c94b027fd3
- https://github.com/futurice/android-best-practices
- https://www.upwork.com/hiring/mobile/tips-and-best-practices-for-android-development/
- http://www.innofied.com/13-android-development-best-practices/
- https://www.tutorialspoint.com/android/android_best_practices.htm

Day 1 Best Practice in Coding, Clean Coding Style + Android

Session 2 Building Better Code Blocks + Android

Links

- https://blog.aritraroy.in/what-my-2-years-of-android-development-have-taught-me-the-hard-way-52b495ba5c51
- **After 2 years of Android App Development, I learnt**
  - Don’t reinvent the wheel
  - Choose Libraries wisely
  - Read More Code
  - Yes, you need ProGuard
  - Use Proper Architecture
  - UI is a joke, if you have to explain it, it means it is bad
    - Learn to design a clean, simple and beautiful interface
  - Analytics is your best friend
  - Be a marketing ninja
  - Write optimized code, code that runs quickly and look out for memory leaks
  - Save more than 5 hours a week with Gradle builds
  - Test, test and when you are done, Test again
  - Android Fragmentation is devil in disguise

- Start using Git, today
- Make it difficult for Hackers
- Develop on a low-end device
-

Additional Links

- https://android-arsenal.com/
- https://www.hackerrank.com/contests/programming-interview-questions/challenges
- https://medium.com/@mmbialas/50-android-studio-tips-tricks-resources-you-should-be-familiar-with-as-an-android-developer-af86e7cf56d2
    - Visual
    - Prevent Android Studio Logcat from clearing the log for the current application when it crashes.
    - Apply a proper code style to your IDE (IntelliJ / Android Studio).
    - Use split screen for increasing efficiency.
    - Distraction Free Mode.
    - Use Live Templates for toasts and if conditions
    - Shortcuts and helpful commands
    - Plugins
    - Resources
    -
- https://stackoverflow.com/documentation/android-studio/2228/android-studio-tips-and-tricks#t=20170924035653643 7156
-

Day 1 Best Practice in Coding, Clean Coding Style + Android

Session 3 Design Tools for Storyboarding, Wireframes + Android

- https://wireframe.cc/
- https://www.hotgloo.com/
- https://proto.io/
- https://moqups.com/
- https://www.invisionapp.com/
- https://androidspeechbook.wordpress.com/
- http://dl.acm.org/citation.cfm?id=2601856
- https://developer.android.com/training/wearables/apps/voice.html
- http://thebooksout.com/downloads/voice-application-development-for-android.pdf
- https://www.packtpub.com/application-development/voice-application-development-android
- https://pdfs.semanticscholar.org/b60a/2e289ce1a844e730f31caea289f224be9655.pdf
- http://www.appsterhq.com/blog/app-development-lifecycle
- http://www.queppelin.com/2016/09/8-phases-of-mobile-app-development-lifecycle/
- https://www.liquidplanner.com/blog/7-tools-to-gather-better-software-requirements/
- https://www.visual-paradigm.com/solution/agiledev/requirements-gathering/
- http://www.softwaretestinghelp.com/5-best-automation-tools-for-testing-android-applications/

Entire App and its source code, other resources

- https://blog.aritraroy.in/20-awesome-open-source-android-apps-to-boost-your-development-skills-b62832cf0fa4
- https://developer.android.com/training/advanced.html
- http://blog.edx.org/15-tips-tricks-android-app-developers-2017
- https://medium.com/@mmbialas/50-android-studio-tips-tricks-resources-you-should-be-familiar-with-as-an-android-developer-af86e7cf56d2
- https://www.codementor.io/codementorteam/5-ways-to-make-learning-android-development-easier-aak4812o3
- https://androidexample.com/?gclid=CjwKCAjwjJjOBRBVEiwAfvnvBLmcOQRsHCQhDC_KNd4okYG1USr_RJq35wLpeHu8DODjtzRBwE2RzxoCVc8QAvD_BwE
- http://www.innofied.com/13-android-development-best-practices/
- https://www.tutorialspoint.com/android/android_best_practices.htm

Videos, Slideshows, PPT

- https://www.youtube.com/watch?v=LpaauWhBzC0
-

Good Sources and Topics to Learn in Android

- https://www.quora.com/What-are-good-sources-for-learning-advanced-Android-development
- https://www.quora.com/How-do-I-learn-advanced-android-programming
- http://www.androidauthority.com/tips-tricks-new-android-app-developers-336322/
- https://www.toptal.com/android/top-10-most-common-android-development-mistakes
- https://dzone.com/articles/few-tips-beginning-android

Books

- https://www.digifloor.com/books-learn-android-application-development-12
- https://bugfender.com/blog/the-best-android-app-development-learning-resources/
- https://www.safaribooksonline.com/library/view/advanced-androidtm-application/9780133892420/
- https://www.digifloor.com/books-learn-android-application-development-12
- https://dzone.com/articles/10-attractive-android
- http://androiddeveloper.galileo.edu/2017/02/08/the-top-10-books-android-programming-2017/
-

Online Courses in Android Programming

- https://www.udemy.com/advance-android-programming-by-9i-technologies/
- https://dzone.com/articles/10-attractive-android

Day 1 Best Practice in Coding, Clean Coding Style + Android

Session 4 Review our example app

- Example 1 – Learning App – To learn Android/ Python/ Kotlin/ iOS-Swift 4
- Features
  - Interactive, rich-content, helpful to developers of all levels
  - Simple, elegant UI
  - Enterprise level Android
  - Available in different devices like laptop, smartphone, wearables, etc.
  -
- Behaviour
  - Supports collaboration with other content contributors, developers, content curators, etc.
  - SaaS delivered app, supports large visitor traffic (up to 100,000 learners, other users, etc.)
  - Has dashboard to support learning analytics
  - Keeps track of time spend by each user and the frequency of visits and the specific content repetitively visited
  -
  -
- Architecture
  - MVP architecture based
  - Components oriented designed
  - Model on MySQL and MongoDB based
  - View kept to minimum essential number of screens and activities
  - Intents, Messages, MessageQueue, IntentResolver, and Handler, Looper, Loader, etc. to considered
  - Threads based efficient design
  - Data Encryption to keep user or visitor data at high privacy

- TDD Environment impact on design and architecture

- **Day 2**
  - **Session 1** **Networking + Android**
  - **Session 2** **Third-Party Libraries – Retrofit, Volley**
    - **Gradle + Android and Third-Party Plug-ins**
  - **Session 3** **Content Provider and Serializing Data Persistence**
    - **REALM database + Android**
  - **Session 4** **Example 1 – Learning App**

Resource **Baiyju's Learning App**

- **Day 3** **Android Testing Strategies**
  - **Session 1** **TDD Environment + Android**
    - Testing Strategies
    - Basic Unit Testing
    - How to Create a Testing Project in Android Studio
    -
  - **Session 2** **Automated Testing Tools – Espresso, Monkey, etc**
  - **Session 3** **NodeJS Server + ExpressJS in local machine, cloud**
  - **Session 4** **MongoDB + MySQL databases in local machine, cloud**

Example 2: Survey or Polling app for People Analytics

Session 1 TDD Environment + Android

- Testing Strategies
- Basic Unit Testing, what to test?

To run your local unit tests, follow these steps:

Be sure your project is synchronized with Gradle by clicking Sync Project in the toolbar.
Run your test in one of the following ways: To run a single test, open the Project window, and then right-click a test and click Run.

- How to Create a Testing Project in Android Studio
  [1] Ensure you have a small app ready and running on Android Studio
  [2] Create your test configurations for the parts in the app you coded into it and run the tests

  - https://developer.android.com/training/testing/unit-testing/local-unit-tests.html
  - https://developer.android.com/studio/test/index.html
  - https://stackoverflow.com/questions/16586409/how-can-i-create-tests-in-android-studio
  - https://io2015codelabs.appspot.com/codelabs/android-studio-testing#1
  - http://evgenii.com/blog/testing-activity-in-android-studio-tutorial-part-1/
  - https://www.toptal.com/android/testing-like-a-true-green-droid
  -
- What is automated testing?
- The popular tools like monkey, espresso, uiautomator, etc.
- Using Espresso

Testing Strategies in Android

Unit Testing is in 2 kinds

1. **Unit Testing**
   JUnit Testing – Set-Up, Coding and Running a Test
   Testing Units of Code without JUnit Testing Automation
   https://io2015codelabs.appspot.com/codelabs/android-studio-testing#1
   http://evgenii.com/blog/testing-activity-in-android-studio-tutorial-part-1/
   https://www.toptal.com/android/testing-like-a-true-green-droid

2. **Instrumented Testing** https://developer.android.com/training/testing/unit-testing/instrumented-unit-tests.html **and** https://developer.android.com/studio/test/index.html

   Instrumented unit tests are tests that run on physical devices and emulators, and they can take advantage of the Android framework APIs and supporting APIs, such as the Android Testing Support Library.

   You should create instrumented unit tests if your tests need access to instrumentation information (such as the target app's **Context**) or if they require the real implementation of an Android framework component (such as a **Parcelable** or **SharedPreferences** object).

   Using instrumented unit tests also helps to reduce the effort required to write and maintain mock code. You are still free to use a mocking framework, if you choose, to simulate any dependency relationships

Set Up Your Testing Environment

In your Android Studio project, you must store the source files for local unit tests at ***module-name***/src/test/java/. This directory already exists when you create a new project.

You also need to configure the testing dependencies for your project to use the standard APIs provided by the JUnit 4 framework. If your test needs to interact with Android dependencies, include the Mockito library to simplify your local unit tests. To learn more about using mock objects in your local unit tests, see Mocking Android dependencies.

In your app's top-level build.gradle file, you need to specify these libraries as dependencies:

```
dependencies {
   // Required -- JUnit 4 framework
   testCompile 'junit:junit:4.12'
   // Optional -- Mockito framework
   testCompile 'org.mockito:mockito-core:1.10.19'
}
```

Create a Local Unit Test Class

Your local unit test class should be written as a JUnit 4 test class.

JUnit is the most popular and widely-used unit testing framework for Java. The latest version of this framework, JUnit 4, allows you to write tests in a cleaner and more flexible way than its predecessor versions. Unlike the previous approach to Android unit testing based on JUnit 3, with JUnit 4, you do not need to extend the junit.framework.TestCase class. You also do not need to prefix your test method name with the 'test' keyword, or use any classes in the junit.framework or junit.extensions package.

To create a basic JUnit 4 test class, create a Java class that contains one or more test methods. A test method begins with the @Test annotation and contains the code to exercise and verify a single functionality in the component that you want to test.

The following example shows how you might implement a local unit test class. The test method emailValidator_CorrectEmailSimple_ReturnsTrueverifies that the isValidEmail() method in the app under test returns the correct result.

```
import org.junit.Test;
import java.util.regex.Pattern;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

public class EmailValidatorTest {

    @Test
    public void emailValidator_CorrectEmailSimple_ReturnsTrue() {
        assertThat(EmailValidator.isValidEmail("name@email.com"), is(true));
    }
    ...
}
```

To test that components in your app return the expected results, use the junit.Assert methods to perform validation checks (or *assertions*) to compare the state of the component under test against some expected value. To make tests more readable, you can use Hamcrest matchers (such as the is() and equalTo() methods) to match the returned result against the expected result.

How to test Intents?

In your **test**, you start by making a call to init() from **Intents**. This will start recording the fired **Intents**, and is required before each **test** where you want to verify **Intent** activity. For each time you call init() , you must follow it up with a call to release() at the end of the **test.**

Testing Intents with Espresso Intents

- http://michaelevans.org/blog/2015/09/15/testing-intents-with-espresso-intents/
- https://gist.github.com/vgonda/1b2520619052cc5bf9b8
- https://collectiveidea.com/blog/archives/2015/06/11/testing-for-android-intents-using-espresso
- https://cate.blog/2016/04/28/testing-intents-on-android-like-stabbing-yourself-in-the-eye-with-a-blunt-implement/
- https://developer.android.com/reference/android/support/test/espresso/intent/Intents.html
- https://developer.android.com/training/testing/espresso/intents.html
-

How can I create Android JUnit test case which tests the content of an Intent generated within an Activity?

Use **ContextWrapper** and override all of the intent functions. Generalizing for all of my Activity tests, I extended the ActivityUnitTestCase class and implemented the solution as a shim.

Alternatively, you could re-factor your code in order to do "clean" unit-test

(I mean a unit test that has everything mocked out except the class under test). Actually, I have a situation myself, where I get a **java.lang.RuntimeException**: Stub! because the code I want to unit test creates new Intents containing mocks that I have injected.

I consider creating my own factory for intents.

Then I could inject a mocked-out factory to my class-under-test

- https://stackoverflow.com/questions/16744737/testing-intent-in-android
- https://developer.android.com/reference/android/support/test/espresso/intent/Intents.html

- https://stackoverflow.com/questions/10343184/how-can-i-unit-test-an-intent-launched-sent-from-an-activity

-

How to test MessageQueue?

- https://developer.android.com/reference/android/os/MessageQueue.IdleHandler.html Callback interface for discovering when a thread is going to block waiting for more messages

- You can retrieve the MessageQueue for the current thread with `Looper.myQueue()`. https://developer.android.com/reference/android/os/MessageQueue.html

- JUnit tests hang on string.contains() in MessageQueue https://stackoverflow.com/questions/33910945/junit-tests-hang-on-string-contains-in-messagequeue

-

How to test Loaders?

Introduced in Android 3.0 (API level 11), the Loader API lets you load data from a content provider or other data source for display in an Activity or Fragment.

If you don't understand why you need the Loader API to perform this seemingly trivial operation, then first consider some of the problems you might encounter without loaders:

- If you fetch the data directly in the **activity or fragment**, your users will suffer from lack of responsiveness due to performing potentially slow queries from the UI thread.
- If you ***fetch the data from*** another thread, perhaps with AsyncTask, then you're responsible for managing both the thread and the UI thread through various activity or fragment lifecycle events, such as **onDestroy()** and configurations changes.

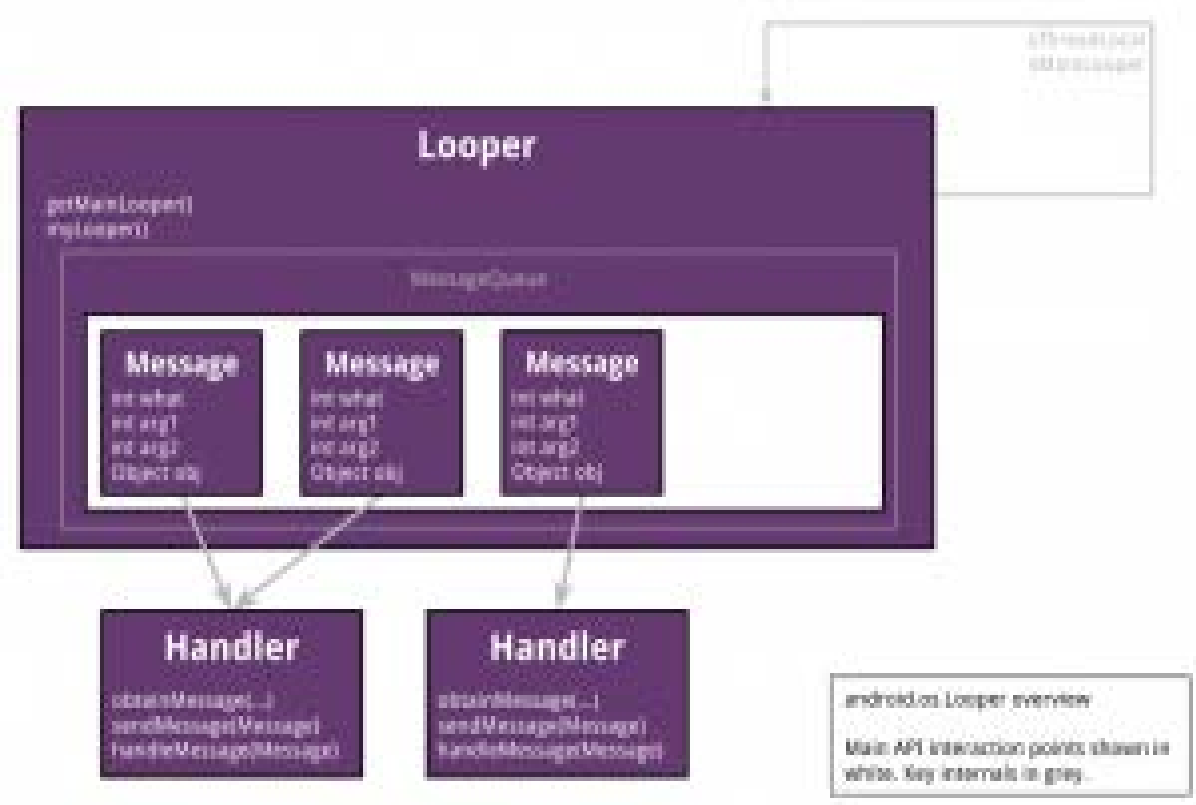
Loaders solve these problems and includes other benefits. For example:

- Loaders run on separate threads to prevent janky or unresponsive UI.
- Loaders simplify thread management by providing callback methods when events occur.
- Loaders persist and cache results across configuration changes to prevent duplicate queries.
- Loaders can implement an observer to monitor for changes in the underlying data source. For example, CursorLoader automatically registers a ContentObserver to trigger a reload when data changes

    - https://developer.android.com/guide/components/loaders.html
    - https://medium.com/google-developers/making-loading-data-on-android-lifecycle-aware-897e12760832
    - http://roisagiv.github.io/blog/2014/01/06/testing-a-loader-using-instrumentation/How to test Handlers? - nice example
    - https://www.grokkingandroid.com/using-loaders-in-android/

How to test Loopers?

There's an interesting low-level class in the Android SDK called Looper. Each app has at least one, because it powers the UI Thread. You can create Android apps perfectly fine without ever using Looper yourself, but it's an interesting thing, so let's take a look under the bonnet.



Structural overview of Android's Looper API.

The visible building blocks are Looper, Handler and Message. Behind the scenes, there's MessageQueue. Each Looper is tied to a single Thread, and each Thread has at most one Looper. The Looper is accessed primarily through the Handler class. Handler has the methods through which you obtain message instances and submit them to the message queue of the associated looper. Once your message is at the front of the queue, the Handler is also the object to process that message. You can implement the message processing behaviour either by subclassing Handler and overriding the handleMessage(..)method, or by creating the Handler with an implementation of the Handler.Callback interface.

Messages have five public fields. The int what defines the message type. It allows a single Handler (or Handler Callback) to handle multiple kinds of messages using a switch construct. The two ints arg1 and arg2 and the object obj are free format arguments that you can use however you see fit. The replyTo field is outside the scope of this blog post.

- http://blog.xebia.com/android-looper-anatomy/
- https://blog.nikitaog.me/2014/10/11/android-looper-handler-handlerthread-i/
-

How to test Handlers?

I use the **android.os.Handler** class to perform tasks on the background.

When unit testing these, I call **Looper.loop()** to make the test thread wait for the background task thread to do its thing.

Later, I call **Looper.myLooper().quit()** (also in the test thread), to allow the test thread to quit the loop and resume the testing logic – ***the result is an issue in logic at runtime***

I also wanted to make a test case for a class that use a Handler. Same as what you did, I use the **Looper.loop()** to have the test thread starts handling the queued messages in the handler.

To stop it, I used the implementation of **MessageQueue.IdleHandler** to notify me when the looper is blocking to wait the next message to come. When it happens, I call the quit() method. But again, same as you I got a problem when I make more than one test case.

- https://stackoverflow.com/questions/3655987/how-to-better-unit-test-looper-and-handler-code-on-android
- https://groups.google.com/forum/#!topic/android-developers/2auUOBrKJkY

I have a class that uses a Handler for a timed, asynchronous activity.
Something like this:

```
public class SampleClass {
  private static final long DELAY = 30000;
  private boolean isRunning = false;
  private Handler handler = new Handler();

  public start() {
    if (!isRunning) {
      isRunning = true;
      handler.post(new Thread(task));
    }
  }
  public stop() {
    isRunning = false;
  }
  private Runnable task = new Runnable() {
    public void run() {
      if (!isRunning) {
        return;
      }
      // Do some tasks
      handler.postDelayed(new Thread(this), DELAY);
    }
  }
}
```

I am trying **to write a unit test** (without having to implement an activity that instantiates the class) but I can't seem to get the items that are posted to the MessageQueue to ever be fired.

Inheriting from **junit.framework.TestCase** **doesn't work**, but then there wouldn't be a MessageQueue for the handler, I'd expect (although, there's no error, but the Runnable never gets called). I tried inheriting the test class from **AndroidTestCase** and **ApplicationTestCase <Application>** but neither of those works, even though the former is supposed to provide a Context and the latter "all the life cycle of an application."

Session 2 Automated Testing Tools – Espresso, Monkey, etc

Expresso Testing – Testing UI of Single App

- https://developer.android.com/training/testing/ui-testing/espresso-testing.html
- https://medium.com/@ali.muzaffar/the-basics-of-unit-and-instrumentation-testing-on-android-7f3790e77bd
- https://www.bloco.io/blog/2015/how-to-test-android-apps-instrumentation-testing
- https://github.com/googlesamples/android-testing
- http://www.vogella.com/tutorials/AndroidTesting/article.html
- http://roisagiv.github.io/blog/2014/01/06/testing-a-loader-using-instrumentation/
- https://collectiveidea.com/blog/archives/2015/06/11/testing-for-android-intents-using-espresso

Testing user interactions within a single app helps to ensure that users do not encounter unexpected results or have a poor experience when interacting with your app. You should get into the habit of creating user interface (UI) tests if you need to verify that the UI of your app is functioning correctly.

The **Espresso testing framework**, provided by the **Android Testing Support Library**, provides APIs for writing UI tests to simulate user interactions within a single target app. Espresso tests can run on devices running Android 2.3.3 (API level 10) and higher. A key benefit of using Espresso is that it provides automatic synchronization of test actions with the UI of the app you are testing. Espresso detects when the main thread is idle, so it is able to run your test commands at the appropriate time, improving the reliability of your tests. This capability also relieves you from having to add any timing workarounds, such as Thread.sleep() in your test code.

The Espresso testing framework is an instrumentation-based API and works with the AndroidJUnitRunner test runner.

Set Up Espresso

Before building your UI test with Espresso, make sure to configure your test source code location and project dependencies, as described in Getting Started with Testing. In the build.gradle file of your Android app module, you must set a dependency reference to the Espresso library:

```
dependencies {
   // Other dependencies ...
   androidTestCompile 'com.android.support.test.espresso:espresso-core:2.2.2'
}
```

Turn off animations on your test device — leaving system animations turned on in the test device might cause unexpected results or may lead your test to fail.

Turn off animations from *Settings* by opening *Developer options* and turning all the following options off:

- **Window animation scale**
- **Transition animation scale**
- **Animator duration scale**

If you want to set up your project to use Espresso features other than what the core API provides, see this resource

- https://stackoverflow.com/questions/6464028/how-to-use-intent-when-testing-android-classes
-

Android UI Instrumentation test with Espresso [Tutorial]

Writing and maintaining a good test suite containing different types of tests requires a good amount of time and discipline on the part of the developer. However, the benefits that comes with doing this includes the following:

- Helps to ensure the correctness of the app features.
- Helps to implement new features with confidence that the new changes will not break existing working code.
- Facilitates building of app features in modules and iterations especially when tests are first written [TDD], then codes are written to validate the test.

Categories of Android unit tests

- **Local unit tests**: These are tests that runs on the JVM. They do not require any native Android library to run. They are found in `app/src/test/java` folder in the **Project perspective**. Unit test should amount to about 60-70% of the test code base.
- **Integration test**: Tests more than one individual module working together that implements a complete functional part of an app. Integration test are located in the `app/src/androidTest/java` folder and requires the Android system to run. Should amount to about 20% of the test code base.

Espresso is a testing framework contained in the Android Testing Support Library. It provides APIs to simulate user interactions and write functional UI tests.

Espresso tests are written based on what user might do while interacting with your app. Basically, you:

- Locate the desired UI element
- Interact with the UI element or check its state.

Android Instrumentation tests are located in the `app/src/androidTest/java` folder and this is the location where you will be writing your tests.

Espresso tests are composed of three major components which are:

- ViewMatchers: These are collection of objects used to find the desired view in the current view hierarchy. They are passed to the `onView` method to locate and return the desired UI element.
- ViewActions: They are used to perform actions such `click` on views. They are passed to the `ViewInteraction.perform()` method.
- ViewAssertions: They are used to assert the state of the currently selected view. They can be passed to the `ViewInteraction.check()` method.

For Example:

```
onView(withId(R.id.my_view))          // withId(R.id.my_view) - ViewMatcher
   .perform(click())                  // click() - ViewAction
   .check(matches(isDisplayed()));    //matches(isDisplayed()) - ViewAssertion
```

Setting up Espresso in your project

- Download the latest Android Support Repository via the SDK Manager or Android Studio
- Open your app's build.gradle file, the one located in `app/build.gradle` and add the following lines inside dependencies:

```
// Android JUnit Runner

androidTestCompile 'com.android.support.test:runner:0.5'
```

```
// JUnit4 Rules
androidTestCompile 'com.android.support.test:rules:0.5'
// Espresso core
androidTestCompile 'com.android.support.test.espresso:espresso-core:2.2.2'
```

- To that same `build.gradle` file, under `android.defaultConfig`, add the following:

```
testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
```

Sample build.gradle file

Set up your device for Espresso tests

To ensure that espresso works as expected on your test device or emulator, turn off animations on your device. Navigate to *Settings -> Developer Options* and turn all the following off under **Drawing**

- Window animation scale
- Transition animation scale
- Animator duration scale

A sample app showing a SignUp and Login screen will be used to demonstrate the basic concepts of Espresso. To follow through with the snippets in the following section, please clone this git repository. `git clone` git@github.com:mayojava/espressoUIIntegrationTest.git

Now let's dive in

The sample code has a main screen with a Login and SignUp button. The first test we will be looking at, is to check that the SignUp screen shows up when the SignUp button is clicked. In the `androidTest` folder, open the `app/src/androidTest/java/.../mainscreen/MainScreenTest.java`file. The first thing to notice here are the annotations:

- `@RunWith(AndroidJUnit4.class)`: Tags the class as an Android JUnit4 class. Espresso unit test should be written as a JUnit 4 test class.
- `@LargeTest`: Qualifies a test to run in >2s execution time, makes use of all platform resources, including external communications. Other test qualifiers are `@SmallTest`and `@MediumTest`.
- `@Rule`: `ActivityTestRule` and `ServiceTestRule` are JUnit rules part of the Android Testing Support Library and they provide more flexibility and reduce the boilerplate code required in tests. These rules provides functional testing of a single activity or service respectively.
- `@Test`: Each functionality to be tested must be annotated with this. The activity under test will be launched before each of the test annotated with `@Test`.

To test that the Signup screen shows up when the Signup button is clicked, we do the following:

1. Locate the Signup button using the `withId` ViewMatcher.
2. Perform a `click()` action on the button.
3. Assert that the Signup screen is displayed.

The code for the steps above is shown in the snippets below:

Next open the `SignUpScreenTest.java` file under `app/src/androidTest/java/.../signup/SignUpScreenTest.java`. Here we want to test that on filling the signup form and clicking the Signup button, the Success screen is displayed with the signup success text, when the signup is complete.

The code for the test above, follow the following steps:

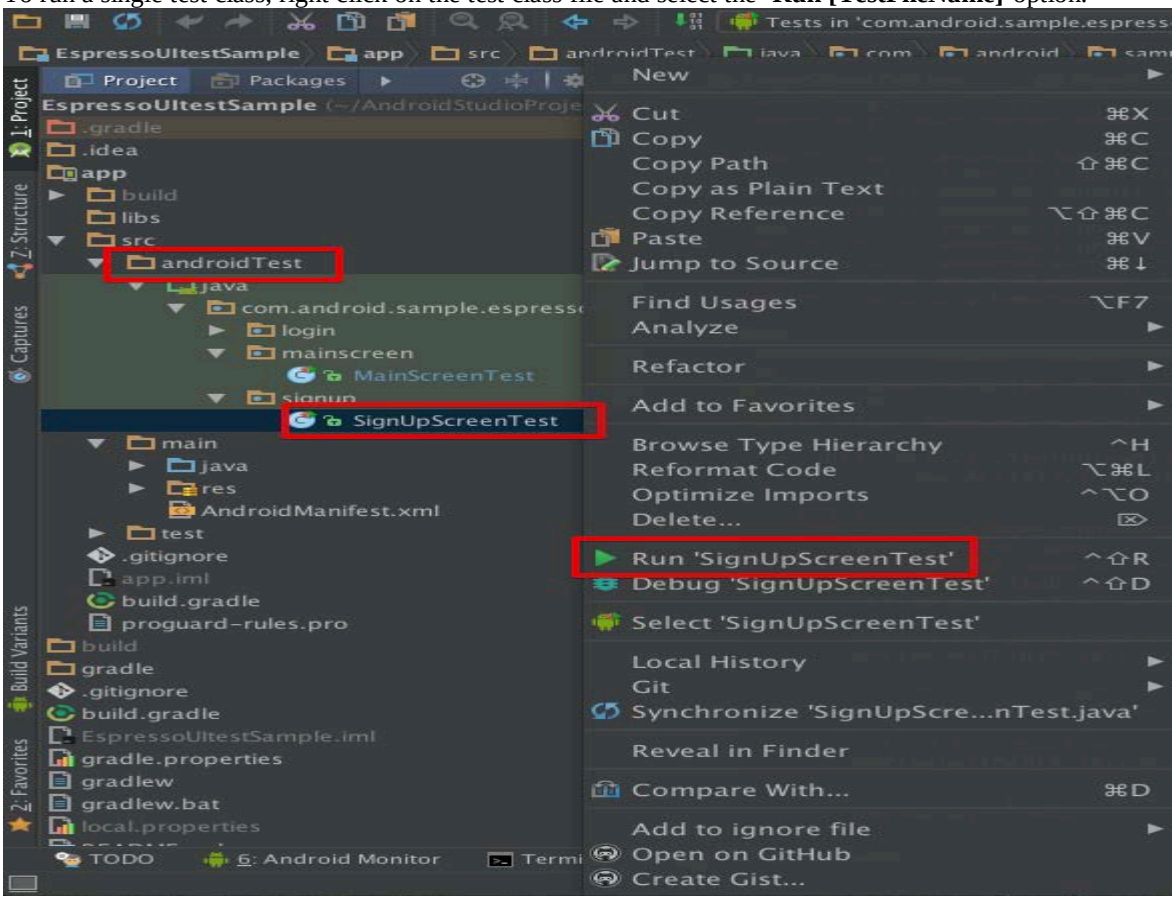
1. Locate each of the sign up form text fields using the `withId` ViewMatcher.
2. Perform `typeText()` action on each, to type in the text for that field.
3. Locate sign up button and perform the `click()` action on the button to submit the form.
4. On sign up completion, assert that the sign up success screen is displayed with the success text.

The code for the steps above is shown below:

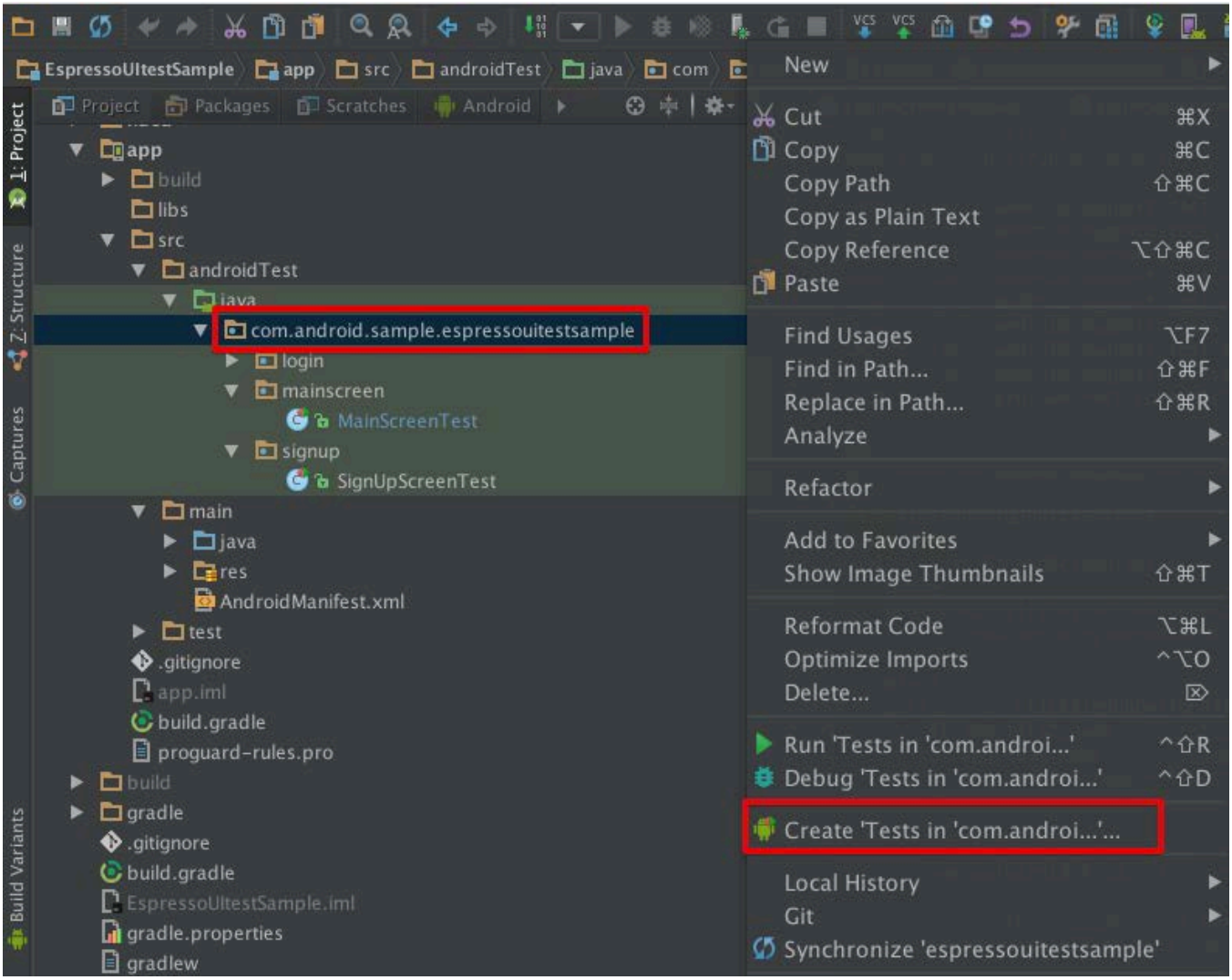
For a list of possible `ViewMatchers`, `ViewActions` and `ViewAssertions` check out the Espresso Cheat Sheet

Running Espresso tests

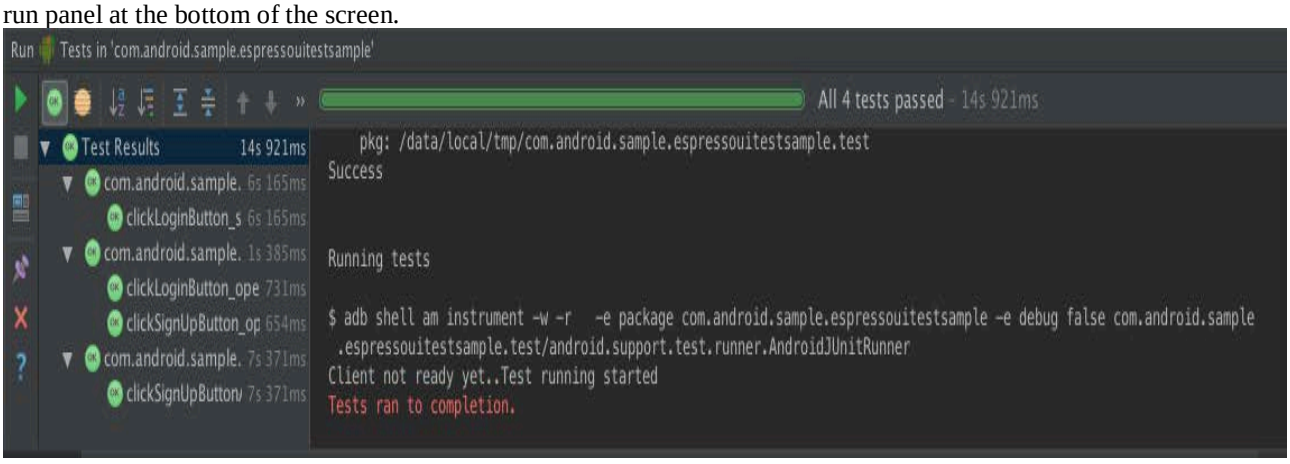
To run a single test class, right click on the test class file and select the **'Run [TestFileName]'**option.



To run many different tests at the same time, you will need to create a build configuration. This can be done easily in Android Studio by right clicking on the package name under `androidTest /java/..` and select **"Create Tests in [package name]"** option.



Select the new run configuration from the run configuration menu and select run. Tests results are displayed in the run panel at the bottom of the screen.



References and further readings:

- Espresso Documentation
- Espresso Intents
- Android Testing Support Library
- Android Testing Codelab
- ViewMatchers
- Hamcrest Matchers
-

The sample code for this post can be found in this github repo.

- https://mayojava.github.io/android/android-ui-instrumentation-test-with-espresso/
- https://teamtreehouse.com/library/testing-in-android
- https://www.coursera.org/learn/engineeringandroidapps/lecture/XQ66A/unit-testing-with-android-studio
-
- https://github.com/codepath/android_guides/wiki/Unit-Testing-with-Robolectric
-
- http://www.vogella.com/tutorials/Mockito/article.html
- https://stackoverflow.com/questions/42188261/how-do-i-unit-test-with-junit-or-mockito-recyclerview-item-clicks
- https://stackoverflow.com/questions/7002843/how-to-create-mock-objects-for-android-activities-junit
-
- https://stackoverflow.com/questions/10207759/unit-testing-for-edittextview-how-do-i-do-it
-
- https://stackoverflow.com/questions/3257856/how-to-populate-a-test-database-in-android?rq=1
-

Mockito + Android

Unit testing is essential to expediting your development time and minimizing regression issues that reach QA or even production. By utilizing architecture patterns like MVP, MVC, MVI, or MVVM I aim to decouple as much business logic from my view as possible, thus allowing me to maximize my unit test coverage.

Today we'll take a look at a few different ways you can improve your Android testing. We'll start with how to utilize Mockito to expedite unit test creation as well as reduce unit test execution speed that rely on Android APIs by utilizing Robolectric. Outside of unit tests it's important to also test the UI to assert current state after a sequence of simulated user gestures or input. This is called functional / instrumentation testing and is achieved by utilizing Espresso. Finally, we'll also cover how to maximize our code reuse for our Espresso tests by adhering to Jake Wharton's Robot Pattern.

Unit testing with Mockito

Mockito allows you to write unit tests that require dependency mocking without all the extra boiler plate code associated with manually creating mock dependencies. Mockito will only help you if you're already using some sort of dependency inversion to abstract the implementation details of your services.

Before Mockito

Before we dive into using Mockito, it's important to consider the following scenario so that you can understand the problem it solves.
In the code below, you can see in my ***LanguagesPresenterTests*** class that I'm manually creating mock dependencies of my **LanguagesPresenter** in order to minimize the code I'm testing to only the **LanguagesPresenter**.
Additionally, I want to test that my presenter retrieves the necessary data from the repository and passes it to the view, the easiest way to achieve this is to maintain a count of method calls in my dependencies.

Mocking Errors

In some cases it might be necessary to override individual mocked methods in order to further test scenarios, an example being error handling. I can easily force a dependency to fail (throw an exception) in order to test my error handling. Below I'm testing to ensure that my view is correctly notified to show the error state when a dependency fails.

Mocking Android with Robolectric

Robolectric mocks the entire Android framework for you. This allows you to run tests that depend on the Android framework under the (test) folder instead of the (**androidTes**t) folder, allow your unit tests to execute much faster.

Remember that unit tests under (**androidTest**) must execute on an actual Android device, which ultimately will slow down unit test execution.

Functional testing with Espresso

Espresso allows you to perform functional/instrumentation testing, in order words allows you to simulate user input and verify UI state and navigation. Espresso tests must live in the (androidTest) folder as your app must actually be deployed and launched in order to simulate user input.

Robot Pattern

The Robot Pattern is a design pattern for writing stable, readable, and maintainable tests. Looking at my two test methods I see some duplicate code, the code that queries for specific UI elements. Now imagine instead of only 2 tests I have 10 tests, the duplicate code is becoming slowly unmaintainable. Yes I could easily resolve this issue by writing a method, but what if my unit tests are split across multiple classes? What if this UI elements are actually part of a fragment that is shared by multiple activities, and I need to test the state of that fragment in each activity? This is where the Robot Pattern comes into play. The Robot Patterns easily allows me to encapsulate common Espresso queries/actions in a robot class.

Mockito Step by Step

We're going to build a (very) simple ordering system. Orders will succeed if there's enough inventory in our warehouse and fail if not. Let's start by creating a very simple little Android application for us to test:

- Create a new Android project in Eclipse called WarehouseManager. Select the "Create a Test Project" option.

- The core abstraction is a warehouse, represented by a Warehouse interface:

```
package com.example;

public interface Warehouse {
boolean hasInventory (String product, int quantity);
void remove (String product, int quantity);
}
```

And here's a very simple concrete implementation of Warehouse:

```
package com.example;

import java.util.HashMap;

public class RealWarehouse implements Warehouse {

    public RealWarehouse () {
        products = new HashMap();
        products.put ("Talisker", 5);
        products.put ("Lagavulin", 2);
    }

    public boolean hasInventory (String product, int quantity) {
        return inStock (product) &gt; = quantity;
    }

    public void remove (String product, int quantity) {
        products.put (product, inStock(product) - quantity);
    }

    private int inStock (String product) {
        Integer quantity = products.get(product);
        return quantity == null ? 0 : quantity;
    }

    private HashMap products;
}
```

1. We remove things from the warehouse by placing an Order:

```
1   package com.example;
2
3   public class Order {
4
5       public Order (String product, int quantity) {
6           this.product = product;
7           this.quantity = quantity;
8       }
9
10      public void fill (Warehouse warehouse) {
11          if (warehouse.hasInventory(product, quantity)) {
12              warehouse.remove(product, quantity);
```

```
13              filled = true;
14            }
15        }
16
17        public boolean isFilled() {
18            return filled;
19        }
20
21        private boolean filled = false;
22        private String product;
23        private int quantity;
24    }
```

2. We'll need a UI to allow us to make orders, so modify main.xml to look like this:

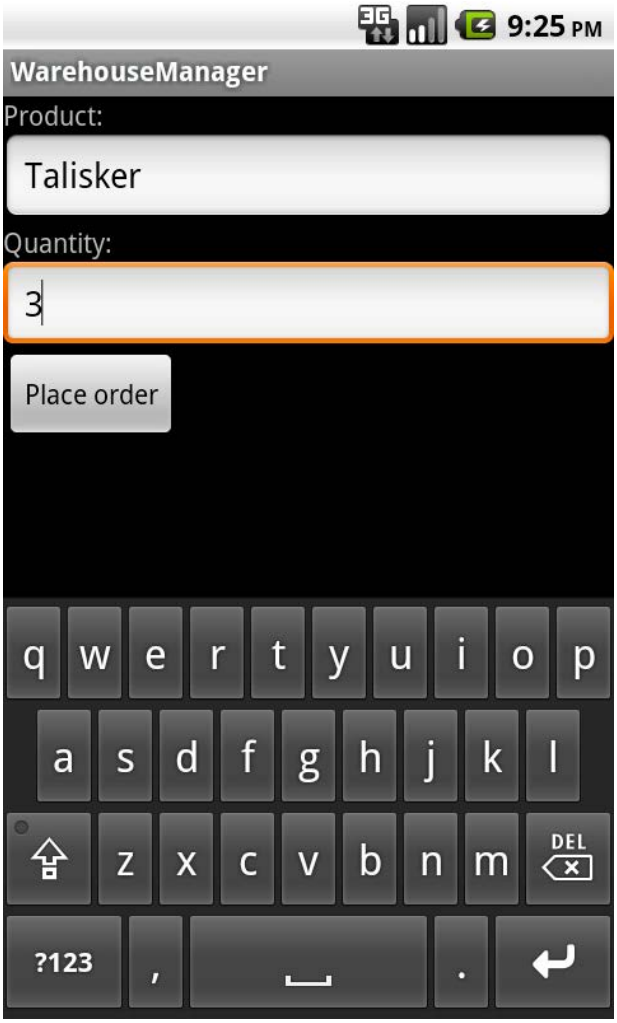
```
        <?xml version="1.0" encoding="utf-8"?>
        <LinearLayout
1           xmlns:android="http://schemas.android.com/apk/res/android"
2           android:orientation="vertical"
3           android:layout_width="fill_parent"
4           android:layout_height="fill_parent"
5           >
6
7       <TextView
              android:layout_width="fill_parent"
9           android:layout_height="wrap_content"
10          android:text="Product:"
11          />
12


        <EditText
15          android:layout_width="fill_parent"
16          android:layout_height="wrap_content"
17          android:id="@+id/product"
18          />
19       <TextView
20          android:layout_width="fill_parent"
21          android:layout_height="wrap_content"
22          android:text="Quantity:"
23          />
24       <EditText
25          android:layout_width="fill_parent"
26          android:layout_height="wrap_content"
27          android:id="@+id/quantity"
28          />
29       <Button
30          android:layout_height="wrap_content"
31          android:layout_width="wrap_content"
32          android:text="Place order"
33          android:onClick="placeOrder" />
        </LinearLayout>
```

3. And finally, here's the implementation of WarehouseManagerActivity:

```
1       package com.example;
2
3       import android.app.Activity;
4       import android.os.Bundle;
5       import android.view.View;
```

```
6   import android.widget.EditText;
7   import android.widget.Toast;
8
9   public class WarehouseManagerActivity extends Activity {
10      /** Called when the activity is first created. */
11      @Override
12      public void onCreate (Bundle savedInstanceState) {
13          super.onCreate (savedInstanceState);
14          setContentView (R.layout.main);
15
16          productEditText = (EditText)findViewById(R.id.product);
17          quantityEditText = (EditText)findViewById(R.id.quantity);
18      }
19
20      public void placeOrder (View view) {
21
22          String product = productEditText.getText().toString();
23          int quantity = Integer.parseInt (quantityEditText.getText().toString());
24          Order order = new Order (product, quantity);
25          order.fill (warehouse);
26
27          String message = order.isFilled () ? "Success" : "Failure";
28          Toast toast = Toast.makeText (this, message, Toast.LENGTH_SHORT);
29          toast.show ();
30      }
31
32
33
34      private Warehouse warehouse = new RealWarehouse ();
35
        private EditText productEditText;
        private EditText quantityEditText;
    }
```

You should now have a little Android application that can be compiled and installed with ant install. Here's what it looks like:



So, now that we've got something to test, let's test it:

- The version of Mockito that supports Android has not yet been released, so you'll need to get the source and build it for yourself. After building, you should have mockito-all-1.9.1-SNAPSHOT.jar in the target directory.
- Create a libs directory underneath **WarehouseManagerTest** and copy mockito-all-1.9.1-SNAPSHOT.jar into it. Download the latest dexmaker into the same directory.
- Finally, we can write our tests, which create mock instances of the Warehouse interface:

```
1   package com.example.test;
2
3   import android.test.InstrumentationTestCase;
4   import com.example.*;
5
6   import static org.mockito.Mockito.*;
7
8   public class OrderTest extends InstrumentationTestCase {
9
10      public void testInStock() {
11          Warehouse mockWarehouse = mock(Warehouse.class);
12
13          when(mockWarehouse.hasInventory("Talisker", 50)).thenReturn(true);
```

```
14
15          Order order = new Order("Talisker", 50);
16          order.fill(mockWarehouse);
17
18          assertTrue(order.isFilled());
19          verify(mockWarehouse).remove("Talisker", 50);
20      }
21
22      public void testOutOfStock() {
23          Warehouse mockWarehouse = mock(Warehouse.class);
24
25          when(mockWarehouse.hasInventory("Talisker", 50)).thenReturn(false);
26
27          Order order = new Order("Talisker", 50);
28          order.fill(mockWarehouse);
29
30          assertFalse(order.isFilled());
31      }
32  }
```

1. Run your tests

References

- http://www.vogella.com/tutorials/Mockito/article.html
- https://developer.android.com/training/testing/unit-testing/local-unit-tests.html
- https://medium.com/square-corner-blog/mockito-on-android-88f84656910
- http://site.mockito.org/
- https://www.grapecity.com/en/blogs/introduction-to-android-testing-using-mockito-and-espresso/
- https://github.com/mockito/mockito
- https://www.grapecity.com/en/blogs/introduction-to-android-testing-using-mockito-and-espresso/
- https://paulbutcher.com/2012/05/15/mockito-on-android-step-by-step/
- https://medium.com/square-corner-blog/mockito-on-android-88f84656910
-
- https://developer.android.com/topic/libraries/testing-support-library/index.html
-

UIAutomator + Android

UI Automator is a UI testing framework suitable for cross-app functional UI testing across system and installed apps. **Note:** This framework requires Android 4.3 (API level 18) or higher. The UI Automator testing framework provides a set of APIs to build UI tests that perform interactions on user apps and system apps. The UI Automator APIs allows you to perform operations such as opening the Settings menu or the app launcher in a test device. The UI Automator testing framework is well-suited for writing black box-style automated tests, where the test code does not rely on internal implementation details of the target app.

The **key features** of the UI Automator testing framework include the following:

- A viewer to inspect layout hierarchy. For more information, see UI Automator Viewer.
- An API to retrieve state information and perform operations on the target device. For more information, see Accessing device state.
- APIs that support cross-app UI testing. For more information, see UI Automator APIs.

UI Automator Viewer - The UIAutomatorViewer tool provides a convenient GUI to scan and analyze the UI components currently displayed on an Android device. You can use this tool to inspect the layout hierarchy and view the properties of UI components that are visible on the foreground of the device. This information lets you create more fine-grained tests using UI Automator, for example by creating a UI selector that matches a specific visible property. The UIAutomatorViewer tool is located in the <android-sdk>/tools/ directory.

Accessing device state

The UI Automator testing framework provides a UiDevice class to access and perform operations on the device on which the target app is running. You can call its methods to access device properties such as current orientation or display size.
The UiDevice class also let you perform actions such as:

- Change the device rotation.
- Press a key or D-pad button.
- Press the Back, Home, or Menu buttons.
- Open the notification shade.
- Take a screenshot of the current window.

For example, to simulate a Home button press, call the UiDevice.pressHome() method.

UI Automator APIs

The UI Automator APIs allow you to write robust tests without needing to know about the implementation details of the app that you are targeting. You can use these APIs to capture and manipulate UI components across multiple apps:

- UiCollection: Enumerates a container's UI elements for the purpose of counting, or targeting sub-elements by their visible text or content-description property.
- UiObject: Represents a UI element that is visible on the device.
- UiScrollable: Provides support for searching for items in a scrollable UI container.
- UiSelector: Represents a query for one or more target UI elements on a device.
- Configurator: Allows you to set key parameters for running UI Automator tests.

A GUI tool to scan and analyse the UI components of an Android application.
The **UIAutomatorViewer** tool provides a convenient visual interface to inspect the layout hierarchy and view the properties of the individual UI components that are displayed on the test device. Using this information, you can later create uiautomator tests with selector objects that target specific UI components to test.
To analyze the UI components of the application that you want to test, perform the following steps after installing the application given in the example.

- Connect your Android device to your development machine
- Open a terminal window and navigate to <android-sdk>/tools/
- Run the tool with this command

References

- https://www.tutorialspoint.com/android/android_ui_testing.htm
- https://developer.android.com/training/testing/ui-testing/uiautomator-testing.html
- https://developer.android.com/training/testing/ui-automator.html
- https://bitbar.com/how-to-get-started-with-ui-automator-2-0/
- https://developer.android.com/training/testing/ui-testing/index.html

Session 3 **NodeJS Server + ExpressJS in local machine, cloud**

Session 4 MongoDB + MySQL databases in local machine, cloud

Android + Databases like MongoDB and MySQL

About MongoDB

MongoDB is a No SQL database. It is an open-source, cross-platform, document-oriented database written in C++. In this database you can insert documents, update documents, delete documents, query documents, projection, sort() and limit() methods, create collection, drop collection etc. MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)

Download, install and set up MongoDB Atlas as a cloud service or DBaaS platform with dashboards optimized to give the best results.

References

- https://www.mongodb.com/download-center#atlas
- https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/
- https://docs.mongodb.com/manual/installation/
- https://www.tutorialspoint.com/mongodb/mongodb_overview.htm

About MongoDB + Android

References

- https://www.tutorialspoint.com/mongodb/
- https://www.guru99.com/mongodb-tutorials.html
- https://docs.mongodb.com/manual/tutorial/
- https://www.javatpoint.com/mongodb-tutorial
- https://www.w3schools.com/nodejs/nodejs_mongodb_create_db.asp
- https://www.tutorialspoint.com/mongodb/
- https://www.guru99.com/mongodb-tutorials.html
- https://docs.mongodb.com/manual/tutorial/
- https://www.javatpoint.com/mongodb-tutorial
-

What is Cloud Computing

Cloud computing means *on demand delivery of IT resources* via the internet with pay-as-you-go pricing. It provides a solution of IT infrastructure in low cost.

Why Cloud Computing?

Actually, Small as well as some large IT companies follows the traditional methods to provide the IT infrastructure. That means **for any IT company, we need a Server Room that is the basic need of IT companies**.
In that server room, there should be a *database server*, *mail server*, *networking*, *firewalls*, *routers*, *modem*, *switches*, *QPS* (Query Per Second means how much queries or load will be handled by the server) , configurable system, *high net speed* and the *maintenance engineers*.
To establish such IT infrastructure, we need to spend lots of money. To overcome all these problems and to reduce the IT infrastructure cost, Cloud Computing comes into existence.

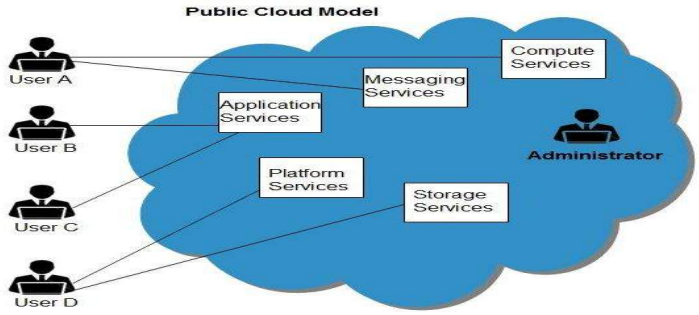
Cloud computing is currently the buzzword in IT industry, and many are curious to know what cloud computing is and how it works. More so because the term CLOUD is intriguing and some people even wonder how do clouds that rain can even remotely be used in Computing.

Cloud Computing can be defined as delivering computing power(CPU, RAM, Network Speeds, Storage OS software) a service over a network (usually on the internet) rather than physically having the computing resources at the customer location.

Example: AWS, Azure, Google Cloud

Cloud Computing provides us means by which we can access the applications as utilities over the internet. It allows us to create, configure, and customize the business applications online.

Public Cloud allows systems and services to be easily accessible to general public. The IT giants such as **Google, Amazon** and **Microsoft** offer cloud services via Internet. The Public Cloud Model is shown in the diagram below.



Distributed cloud computing

Distributed cloud is the application of **cloud computing** technologies to interconnect data and applications served from multiple geographic locations. **Distributed**, in an information technology (IT) context, means that something is shared among multiple systems which may also be in different locations.

Another cloud computing model that may be useful in many cases is to have many micro or even nano data centers, e.g., ISP POPs, interconnected by medium to high bandwidth links, and to manage the data centers and interconnecting links as if they were one larger data center. This distributed cloud model is perhaps a better match for private enterprise clouds, which tend to be smaller than the large, public mega data centers, but it has attractions for public clouds run by telecom carriers which have facilities in various cities with power, cooling, and bandwidth already available. It is also attractive for mobile operators, since it provides a platform whereby applications can be deployed and easily managed that could benefit from a tighter coupling to the wireless access network. Finally, research testbeds like GENI and FIRE are evolving towards the distributed cloud model. The two models aren't mutually exclusive; a public cloud operator with many large data centers distributed internationally could manage their network of data centers like a distributed cloud.

Distributed clouds also encompasses federated clouds, where data centers managed by different organizations federate to allow users to utilize any of the data centers. The distinction between federated clouds and the model of more tightly coupled distributed clouds is whether authentication is handled centrally or whether each data center handles authentication individually, with authentication for entry into the distributed cloud implemented using single sign-on. Additionally, the more tightly coupled distributed cloud model may hide the locality distinctions between physical data centers and present the distributed cloud as one single data center without exposing the network interconnections. In the latter model, orchestration software manages the user's view of the compute/storage/networking resources to hide locality. Such a model may be important in cases where locality isn't an important characteristic for application deployment. In other cases, locality may be important and then it will be exposed through the orchestration layer.

Many applications can benefit from distributed cloud. Applications that can benefit from locality include real time applications where latency is important such as smart grid for control of the electrical network and any application where the local regulatory environment requires user account data to be stored in the same country where the user lives. Hybrid cloud applications, where a private data center delegates peak processing to a public data center, allow enterprises to provision their data centers for average rather than peak load, thereby saving CAPEX. Content delivery networks are another example where positioning of data close to the user is required. The opposite case – moving the processing closer to the data – may become important where the data sets are large and networking costs prohibit actually moving the data. Last but not least, in many cooperative scenarios such as Internet-based commerce or advanced manufacturing, different stakeholders or users have existing data and/or system deployments in distinct data centers – public or private – which need to interact without the ability of migrating components ("distribution-by-need" as opposed to "distribution-by-design").

The vision behind distributed cloud is to utilize software as a means to aggregate compute/storage/networking resources across distributed physical data centers, with scalability and reliability automated by scale-out, primarily software based solutions, and with locality exposed as a deployment criterion for those applications which require it. The scale out model of service deployment – deploying many small instances of a service to meet demand rather than a few large instances – has proven successful for IaaS and SaaS, distributed cloud applies the same scale out model to data centers.

The essential difference is that cloud computing is about infrastructure while distributed system is about execution and communication between processes. Let me explain:

- Cloud is about elasticity, multi-tenancy et al[1]. Basically on-demand compute/storage. (I do disagree with Tony Li - cloud is not about CPU and it is disruptive with real value - not just marketing)
- Cloud is how you procure your compute & storage. BTW, it could be a SaaS where one also procures the processing (like CRM) [2]

- A distributed system is where the compute is dome across multiple processes - in the same computer or across different computer systems. Distributed systems can run in a cloud infrastructure as well.
- Usually distributed processing is layered over some form of IPC (Inter Process Communication) Protocol - CORBA,DCOM et al in older days and now REST, Akka, JNI, et al
- Also the distributed processing can be compute parallelism or data parallelism

[1] Six essential traits of an Enterprise Cloud Infrastructure or how to define a Cloud without defining it
[2] A Simple Minded Cloud Reference Architecture - Part II

Distributed computing comprises of multiple software components that belong to multiple computers. The system works or runs as a single system. The computers forming the distributed architecture may or may not be closely located. These systems are often preferred for the scalability factor. It is quite easy to add new components to the system without disturbing the existing system.

References

- https://www.lynda.com/Cloud-Computing-training-tutorials/1385-0.html
- https://www.lynda.com/IT-Infrastructure-tutorials/Learning-Cloud-Computing-Serverless-Computing/599623-2.html
- https://www.lynda.com/IT-Infrastructure-tutorials/Cloud-Architecture-Core-Concepts/599615-2.html
- https://www.cloudcomputing-news.net/news/2015/mar/05/analysing-rise-distributed-cloud-model/
-
- https://www.lynda.com/Cloud-Computing-training-tutorials/1385-0.html
- https://www.guru99.com/cloud-computing-for-beginners.html
-
-
- http://infocom2017.ieee-infocom.org/workshop/dcc-distributed-cloud-computing-applying-scale-out-data-center
- https://www.dezyre.com/article/cloud-computing-vs-distributed-computing/94

Big Data, Distributed Big Data and Big Data Analysis

Big Data is extremely large data sets that may be analyzed computationally to reveal patterns, trends, and associations, especially relating to human behaviour and interactions.

Big data is a term for **data sets** that are so large or complex that traditional data processing application software is inadequate to deal with them. Big data challenges include capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating and information privacy.

Lately, the term "big data" tends to refer to the use of **predictive analytics**, **user behavior analytics**, or certain other advanced data analytics methods that extract value from data, and seldom to a particular size of data set.

"There is little doubt that the quantities of ***data now available are indeed large***, but that's not the most relevant characteristic of this new data ecosystem." Analysis of data sets can find new correlations to "spot business trends, prevent diseases, combat crime and so on."

Scientists, business executives, practitioners of medicine, advertising and governments alike regularly meet difficulties with large data-sets in areas including Internet search, fintech, urban informatics, and business informatics.

Scientists encounter limitations in e-Science work, including meteorology, genomics, connectomics, complex physics simulations, biology and environmental research.

Data sets grow rapidly - in part because they are increasingly gathered by cheap and numerous information-sensing Internet of things devices such as mobile devices, aerial (remote sensing), software logs, cameras, microphones, radio-frequency identification (RFID) readers and wireless sensor networks.

The world's technological per-capita capacity to store information has roughly doubled every 40 months since the 1980s; as of 2012, every day 2.5 exabytes (2.5×10^{18}) of data are generated.

One question for large enterprises is determining who should own big-data initiatives that affect the entire organization.

Big data is a term that describes the large volume of data – both structured and unstructured – that inundates a business on a day-to-day basis. But it's not the amount of data that's important. It's what organizations do with the data that matters. Big data can be analyzed for insights that lead to better decisions and strategic business moves. While the term "big data" is relatively new, the act of gathering and storing large amounts of information for eventual analysis is ages old.

The concept gained momentum in the early 2000s when industry analyst Doug Laney articulated the now-mainstream definition of big data as the three Vs:

Volume. Organizations collect data from a variety of sources, including business transactions, social media and information from sensor or machine-to-machine data. In the past, storing it would've been a problem – but new technologies (such as Hadoop) have eased the burden.

Velocity. Data streams in at an unprecedented speed and must be dealt with in a timely manner. RFID tags, sensors and smart metering are driving the need to deal with torrents of data in near-real time.

Variety. Data comes in all types of formats – from structured, numeric data in traditional databases to unstructured text documents, email, video, audio, stock ticker data and financial transactions.

At SAS, we consider two additional dimensions when it comes to big data:

Variability. In addition to the increasing velocities and varieties of data, data flows can be highly inconsistent with periodic peaks. Is something trending in social media? Daily, seasonal and event-triggered peak data loads can be challenging to manage. Even more so with unstructured data.

Complexity. Today's data comes from multiple sources, which makes it difficult to link, match, cleanse and transform data across systems. However, it's necessary to connect and correlate relationships, hierarchies and multiple data linkages or your data can quickly spiral out of control.

Diagram 1

Diagram -2

Distributed Big Data

The highly centralized enterprise data center is becoming a thing of the past, as organizations must embrace a more distributed model to deal with everything from content management to big data. Here we examine how technologies like Hadoop and NoSQL fit into modern distributed architectures in a way that solves scalability and performance problems.

1. **Big data is distributed data**

Big data is a nebulous term with many different definitions. The key thing to remember is that in this day and age, big data is distributed data. This means the data is so massive it cannot be stored or processed by a single node.

The days of buying a single big iron server from IBM or Sun to handle all your business intelligence needs are long gone. It's been proven by Google, Amazon, Facebook, and others that the way to scale fast and affordably is to use commodity hardware to distribute the storage and processing of our massive data streams across several nodes, adding and removing nodes as needed.

2. **You're going to hear the words "Hadoop" and "MapReduce"**

What is Hadoop? It is an open source platform for consolidating, combining and understanding large-scale data in order to make better business decisions. Hadoop is the technology powering many (but not all) big data analytics infrastructures.

There are 2 key parts to Hadoop:

- HDFS (Hadoop distributed file system) which lets you store data across multiple nodes.
- MapReduce which lets you process data in parallel across multiple nodes.

Although Hadoop is one of the most popular solutions for crunching big data — there are plenty others. Big data can't be shoehorned into one flavor of technology. The important characteristic is that you're able to draw insights from large quantities of data, independent of specific technologies.

3. **You *can* understand MapReduce without a degree from Stanford**

The best plain English explanation of MapReduce I've encountered (paraphrasing): We want to count all the books in the library. You count up shelf #1. I count up shelf #2. That's map.Now we get together and add our individual counts. That's reduce. For a deeper understanding, Wikipedia is a good place to start.

4. **Distributed data generation is fueling big data growth**

The reason we have data problems so big that we need large-scale distributed computing architecture to solve is that the creation of the data is also large-scale and distributed. Most of us walk around carrying devices that are constantly pulsing all sorts of data into the cloud and beyond – our locations, our photos, our tweets, our status updates, our connections, even our *heartbeats*. For every human-generated piece of data there's likely associated machine-generated data. And then there's the metadata. The data is abundant and it's extremely valuable.

5. **Machine learning is…awesome!**

One of the key differentiators in big data analytics are the machine learning algorithms used to answer interesting questions and derive value from the 0s and 1s we're furiously chewing up and spitting back out.

References

- https://www.internet2.edu/communities-groups/advanced-networking-groups/distributed-big-data-analytics/
- http://www.theserverside.com/feature/How-big-data-and-distributed-systems-solve-traditional-scalability-problems
- https://blog.varonis.com/5-things-you-should-know-about-big-data/

Big Data Analytics

Big data analytics is the process of examining **large** and varied **data** sets -- i.e., **big data** -- to uncover hidden patterns, unknown correlations, market trends, customer preferences and other useful information that can help organizations make more-informed business decisions.

Big Data is being generated at all times. **Every digital process** and social media exchange produces it. Systems, sensors and mobile devices transmit it. Much of this data is coming to us in an unstructured form, making it difficult to put into structured tables with rows and columns. To **extract insights** from this complex data, Big Data projects often rely on **cutting edge analytics** involving data science and machine learning. Computers running sophisticated algorithms can help enhance the veracity of information by sifting through the noise created by Big Data's massive volume, variety, and velocity.

Big data analytics is the process of examining large and varied data sets -- i.e., big data -- to uncover hidden patterns, unknown correlations, market trends, customer preferences and other useful information that can help organizations make more-informed business decisions.

Big data analytics examines large amounts of data to uncover hidden patterns, correlations and other insights. With today's technology, it's possible to analyze your data and get answers from it almost immediately – an effort that's slower and less efficient with more traditional business intelligence solutions.

Dealing with unstructured and structured data, **Data Science** is a field that encompasses anything related to data cleansing, preparation, and analysis. Put simply, Data Science is an umbrella term for techniques used when trying to extract insights and information from data.

Big Data analytics is the process of collecting, organizing and analyzing large sets of data (*called* Big Data) to discover patterns and other useful information. Big Data analytics can help organizations **to better understand the information contained within the data and will also help identify the data that is most important to the business** and future business decisions. Analysts working with Big Data typically want the *knowledge* that comes from analyzing the data.

Example data from construction business segment when using BIM, Revit, Navisworks on large projects – 3D models, data sets in document form like spreadsheets, word, pdf, etc. and data from collaboration work online

References

- https://www.tutorialspoint.com/big_data_analytics/
- https://www.greatlearning.in/pg-program-big-data-analytics/
- https://www.ibm.com/analytics/us/en/big-data/
- https://www.oracle.com/big-data/index.html
- http://searchbusinessanalytics.techtarget.com/definition/big-data-analytics
-
-
- https://www.sas.com/en_us/insights/big-data/what-is-big-data.html
- http://searchcloudcomputing.techtarget.com/definition/big-data-Big-Data
-
-

Day 4 Disruptive Technologies + Android

- **Session 1 Android + Cloud (AWS)**
  - Option 1 - Getting started with AWS > select platform 'Android' >
    - One method – is to download the Amazon AWS SDK
    - Unzip it
    - Add the .jar files as Libraries using Android Studio
    -
  - Option 2 – https://www.studytutorial.in/android-application-integrate-to-amazon-s3-tutorial
    - Login to Amazon Cognito and click on the **Manage federated identities** button
    -
  -

References

- https://aws.amazon.com/developers/getting-started/
- http://docs.aws.amazon.com/mobile/sdkforandroid/developerguide/setup.html
- http://docs.aws.amazon.com/mobile/sdkforandroid/developerguide/aws-android-dg.pdf
- https://github.com/awsdocs/aws-android-developer-guide/blob/master/doc_source/setup.rst
- https://www.youtube.com/watch?v=kJap3OTbFuY
-
- https://aws.amazon.com/products/developer-tools/
- https://aws.amazon.com/ec2/developer-resources/
- https://aws.amazon.com/tools/

- https://aws.amazon.com/certification/certified-developer-associate/
- https://aws.amazon.com/training/course-descriptions/developing/
- https://aws.amazon.com/api-gateway/?sc_channel=PS&sc_campaign=acquisition_IN&sc_publisher=google&sc_medium=api_gateway_b&sc_content=sdk_bmm&sc_detail=%2Baws%20%2Bsdk&sc_category=api_gateway&sc_segment=186799005728&sc_matchtype=b&sc_country=IN&s_kwcid=AL!4422!3!186799005728!b!!g!!%2Baws%20%2Bsdk&ef_id=WOr8xwAABIeuJQXd:20170925031055:s

- https://aws.amazon.com/net/
- https://aws.amazon.com/documentation/
- http://docs.aws.amazon.com/mobile/sdkforandroid/developerguide/aws-android-dg.pdf
-
- https://www.studytutorial.in/android-application-integrate-to-amazon-s3-tutorial

Prerequisites

Before you can use the |sdk-android|, you will need the following:

- An AWS Account

- Android 2.3.3 (API Level 10) or higher (for more information about the Android platform, see Android Developers)
- Android Studio or Android Development Tools for Eclipse

After completing the prerequisites, you will need to do the following to get started:

1. Get the |sdk-android|.
2. Set permissions in your :file:`AndroidManifest.xml` file.

3. Obtain AWS credentials using Amazon Cognito.

Step 1: Get the |sdk-android|

There are three ways to get the |sdk-android|.

Option 1: Using Gradle with Android Studio

If you are using Android Studio, add the :file:`aws-android-sdk-core` dependency to your :file:`app/build.gradle` file, along with the dependencies for the individual services that your project will use, as shown below.

```
dependencies {
   compile 'com.amazonaws:aws-android-sdk-core:2.2.+'
   compile 'com.amazonaws:aws-android-sdk-s3:2.2.+'
   compile 'com.amazonaws:aws-android-sdk-ddb:2.2.+'
}
```

- https://www.sitepoint.com/quick-tip-what-is-gradle-and-how-does-it-work-with-android-studio/
- https://developer.android.com/studio/write/sample-code.html
- https://developer.android.com/samples/index.html
- https://developer.android.com/studio/releases/gradle-plugin.html
- http://www.c-sharpcorner.com/blogs/solving-gradle-sync-issues-in-android-studio-23-update
- https://developer.android.com/studio/build/index.html
- https://stackoverflow.com/questions/42603047/after-updating-to-android-studio-2-3-build-gradle-is-not-able-to-build
- https://developer.android.com/studio/intro/keyboard-shortcuts.html
- https://developer.android.com/studio/build/index.html#settings-file
-

Option 2:

- aws-android-sdk-2.6.2.zip
-

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. With a few clicks in the AWS Management Console, you can create an API that acts as a “front door” for applications to access data, business logic, or functionality from your back-end services, such as workloads running on Amazon Elastic Compute Cloud (Amazon EC2), code running on AWS Lambda, or any Web application. Amazon API Gateway handles all the tasks involved in accepting and processing

up to hundreds of thousands of concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management. Amazon API Gateway has no minimum fees or startup costs. You pay only for the API calls you receive and the amount of data transferred out.

- **Session 2 Android + Big Data (MongoDB)**
- **Session 3 Different SDKs for Android**
- **Session 4 Example 3 Catering Services, order food online**

Day 5

- **Session 1** — **Advanced UI in Android**
  - Lists, Cards, Colors, RecycleView
  - Material Design, Material Design with the Android Design Support Library
  - Responsive Images
  - Sample gallery of android UI design patterns, templates and themes
  - Review of learning done in day 1, session 2 and at this point in time
  -
  -
- **Session 2** — **Agile Development and Android**
  - What is Agile, Agility, Simplicity, etc.
  - The different methodologies in Agile Development – XP, Scrum, Crystal, FDD, etc.
  - Terms, Concepts, Adopting XP,
  - Practicing XP – Thinking, Collaborating, Releasing, Planning, Developing, etc.
  - Mastering Agility – Values and Principles, Eliminate Waste, Improve Process, etc.
  -
- **Session 3** — **Design Patterns in Android**
  - MVC
  - MVP
  - MVVM
  - Singleton, use of a single object, single instance with a class
  - Consumer-Producer with threads
  -
- **Session 4** — **Reactive Programming and Clean Architecture**
  - Reactive Programming, Rx, RxJava, RxAndroid
  - Clean Architecture and Android
    - o

Session 1 Advanced UI in Android

Android UI – Lists and Cards

The RecyclerView widget is a more advanced and flexible version of ListView. This widget is a container for displaying large data sets that can be scrolled very efficiently by maintaining a limited number of views. Use the RecyclerView widget when you have data collections whose elements change at runtime based on user action or network events.

The RecyclerView class simplifies the display and handling of large data sets by providing:

- Layout managers for positioning items
- Default animations for common item operations, such as removal or addition of items

You also have the flexibility to define custom layout managers and animations for RecyclerView widgets.

Figure 1. The `RecyclerView` widget.

To use the RecyclerView widget, you have to specify an adapter and a layout manager. To create an adapter, extend the RecyclerView.Adapter class. The details of the implementation depend on the specifics of your dataset and the type of views. For more information, see the examples below.

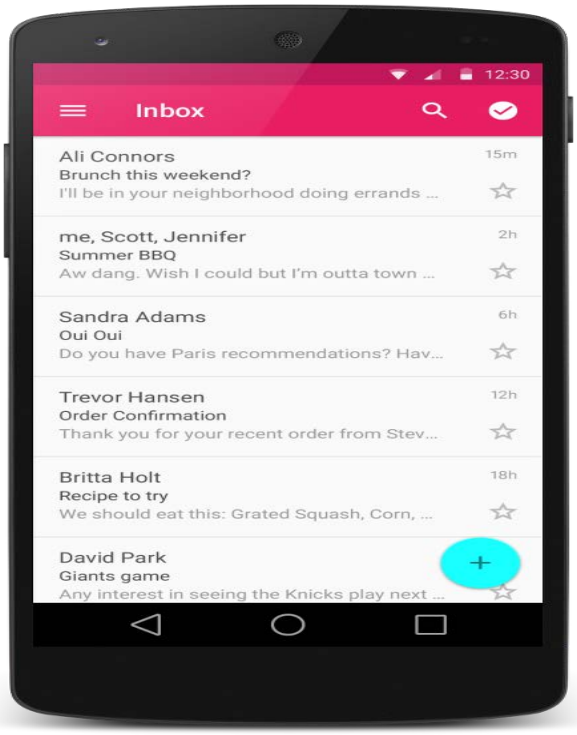


Figure 2 - Lists with `RecyclerView`.

A **layout manager** positions item views inside a RecyclerView and determines when to reuse item views that are no longer visible to the user. To reuse (or *recycle*) a view, a layout manager may ask the adapter to replace the contents

of the view with a different element from the dataset. Recycling views in this manner improves performance by avoiding the creation of unnecessary views or performing expensive findViewById() lookups.

RecyclerView provides these built-in layout managers:

- LinearLayoutManager shows items in a vertical or horizontal scrolling list.
- GridLayoutManager shows items in a grid.
- StaggeredGridLayoutManager shows items in a staggered grid.

To create a custom layout manager, extend the RecyclerView.LayoutManager class.

Animations

Animations for adding and removing items are enabled by default in RecyclerView. To customize these animations, extend the RecyclerView.ItemAnimator class and use the RecyclerView.setItemAnimator()method

Create Cards

CardView extends the FrameLayout class and lets you show information inside cards that have a consistent look across the platform. CardView widgets can have shadows and rounded corners.
To create a card with a shadow, use the card_view:cardElevation attribute. CardView uses real elevation and dynamic shadows on Android 5.0 (API level 21) and above and falls back to a programmatic shadow implementation on earlier versions. For more information, see Maintaining Compatibility.

Use these properties to customize the appearance of the CardView widget:

- To set the corner radius in your layouts, use the card_view:cardCornerRadius attribute.
- To set the corner radius in your code, use the CardView.setRadius method.
- To set the background color of a card, use the card_view:cardBackgroundColor attribute.

The following code example shows you how to include a CardView widget in your layout: main.xml file

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   xmlns:card_view="http://schemas.android.com/apk/res-auto"
   ... >
   <!-- A CardView that contains a TextView -->
   <android.support.v7.widget.CardView
      xmlns:card_view="http://schemas.android.com/apk/res-auto"
      android:id="@+id/card_view"        android:layout_gravity="center"
      android:layout_width="200dp"        android:layout_height="200dp"      card_view:cardCornerRadius="4dp">

      <TextView            android:id="@+id/info_text"            android:layout_width="match_parent"
         android:layout_height="match_parent" />

   </android.support.v7.widget.CardView>
</LinearLayout>
```

Add Dependencies

The RecyclerView and CardView widgets are part of the v7 Support Libraries. To use these widgets in your project, add these Gradle dependencies to your app's module:

```
dependencies {
   ...
```

```
    compile 'com.android.support:cardview-v7:21.0.+'
    compile 'com.android.support:recyclerview-v7:21.0.+'
}
```

Material Design – Lists, Cards and Colors

Links or References

- https://developer.android.com/training/material/lists-cards.html
- https://developer.android.com/training/material/lists-cards.html#CardView
- https://www.androidhive.info/2016/05/android-working-with-card-view-and-recycler-view/ **sample code**
- http://www.truiton.com/2015/03/android-cardview-example/
- https://code.tutsplus.com/tutorials/getting-started-with-recyclerview-and-cardview-on-android--cms-23465
- https://material.io/guidelines/components/cards.html
- https://google-developer-training.gitbooks.io/android-developer-fundamentals-course-practicals/content/en/Unit%202/52_p_material_design_cards_and_the_fab.html
-

Firebase is Google's mobile platform that helps you quickly develop high-quality apps and grow your business.

- Set Up Environment with Android Studio and Nox Emulator
- View-Pager Activity/ Image Recycler View
- Flavours/ EventBus (Using an event Bus is necessary to become a better android developer. We will learn what an event bus is and how to implement it into our project.)
- Activity Dialog / Recycler-View Swipe
- Advanced Recycler-View
- YouTube API/Picture SlideShow
- Expandable Recycler-View
- Google Maps API/PDF WebView
- Googles Firebase
- Styling the App/Publishing

References

- https://developer.android.com/training/best-ui.html
- https://www.sitepoint.com/the-11-best-tutorials-on-mobile-design/
- https://androidexample.com/?gclid=EAIaIQobChMIsJWS3anX1gIV0I5oCh3gqAW2EAMYASAAEgKnS_D_BwE
-
- https://www.codementor.io/lintonye/android-ui-tutorial-layouts-and-animations-85jkhcblt
- https://www.udemy.com/build-an-advance-android-app/
- https://developer.android.com/guide/topics/ui/index.html
- https:/ https://stackoverflow.com/questions/7781795/how-to-implement-advanced-ui-designs-in-android/developer.android.com/training/best-ui.html
- http://www.viralandroid.com/2015/11/android-user-interface-ui-design-tutorial.html
-
- https://firebase.google.com/?utm_source=google&utm_medium=cpc&utm_campaign=1001467%20%7C%20Firebase*%20Brand%20GENERIC%20%7C%20India%20%7C%20en%20%7C%20Desk%2BTab%2BMobile%20%7C%20Text%20%7C%20BKWS%20%5B2017%5D&utm_term=%7Bkeyword%7D&gclid=EAIaIQobChMItKCQ6KvX1gIV1oVoCh1lfwF0EAAYASAAEgJVtPD_BwE

- https://firebase.google.com/

Material Design, a new design language that gives design guidelines for Android apps and apps on other platforms, was introduced with the release of Android 5.0 Lollipop.

With it came new UI components such as the 'Floating Action Button'. Implementing these new components while ensuring backward compatibility was typically a tedious process. Third party libraries would usually be needed to streamline the process.

At this year's Google IO conference, Google introduced the Android Design Support Library which brings a number of important material design components to developers. The components are backwards compatible, with support back to Android 2.1 and implementing them is easier than previously. The library includes a navigation drawer view, floating labels for editing text, a floating action button, snackbar, tabs and a motion and scroll framework to tie them together. In this tutorial, we'll create an app that showcases these components.
https://www.sitepoint.com/material-design-android-design-support-library/

- https://www.sitepoint.com/the-11-best-tutorials-on-mobile-design/
-
-

Android Layout and Animations

Layouts

- FrameLayout
- LinearLayout
- RelativeLayout
- GridLayout

The basic building block for Android UI is layout. So we're going to see how we can actually build those layouts. There are a few options like frame layouts, linear layouts relative layouts, grid layouts. The top three (frame layouts, linear layouts relative layouts) there are probably the most important ones and we're going to see how we use it in practice.

Views

Types of views:

- TextView, ImageView, Button...
- ListView, GridView

- Recycler View
- ViewPager
-

Before we put all these things inside a grid, let's have an overview of what we have here:
You can have your **custom views** written in Java already. A **ListView** shows things in a vertical list, which is one of the most used views in Android. Furthermore, **GridView** is kind of outdated.

Since last summer, Google actually introduced the **RecyclerView**. If you're writing new projects, use **RecyclerView** instead of a **ListView** because it's a lot more flexible and it gives you a lot of other options. For one thing, **ListView** will only show you vertical lists. To create a horizontally scrolling list, you're going to have to find a third-party library or just code it all by yourself. However, if you use a **RecyclerView**, you can do pretty much anything.

For our grid here, we're going to use a **RecyclerView** as well. But the idea is that you can actually populate these **RecyclerViews** and then see it on the real device using this tool.

- https://www.codementor.io/lintonye/android-ui-tutorial-layouts-and-animations-85jkhcblt
- https://www.codementor.io/lintonye/android-ui-tutorial-layouts-and-animations-85jkhcblt
-
-

Session 2 Agile Development and Android

"**Agile Development**" is an umbrella term for several iterative and incremental software **development** methodologies. The most popular **agile** methodologies include Extreme **Programming** (XP), Scrum, Crystal, Dynamic Systems **Development** Method (DSDM), Lean **Development**, and Feature-Driven **Development** (FDD)

Agile software **development** describes a set of values and principles for software **development** under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams.

Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product. Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks.

Every iteration involves cross functional teams working simultaneously on various areas like −

- Planning
- Requirements Analysis
- Design
- Coding
- Unit Testing and
- Acceptance Testing.

At the end of the iteration, a working product is displayed to the customer and important stakeholders.

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Twelve principles behind the Agile Manifesto

We follow these principles:

1. Our ***highest priority*** is to satisfy the customer through early and continuous delivery of valuable software.
2. ***Welcome changing requirements***, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, ***with a preference to the shorter timescale.***
4. Business people and developers must ***work together daily throughout*** the project.
5. Build ***projects around motivated individuals***. Give them the environment and support they need, and trust them to get the job done.
6. ***The most efficient and effective method of conveying information*** to and within a development team is face-to-face conversation.
7. Working software ***is the primary measure of progress***.
8. Agile processes ***promote sustainable development***. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. ***Continuous attention to technical excellence*** and good design enhances agility.
10. **Simplicity**--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from ***self-organizing teams***.
12. At regular intervals, ***the team reflects on how to become more effective***, then tunes and adjusts its behavior accordingly.

Please click here for further detail on Agile Manifesto.

The various agile Scrum methodologies share much of the same philosophy, as well as many of the same characteristics and practices. But from an implementation standpoint, each has its own recipe of practices, terminology, and tactics. Here we have summarized a few of the main agile software development methodology contenders:

Agile Scrum Methodology

Scrum is a lightweight agile project management framework with broad applicability for managing and controlling iterative and incremental projects of all types. Ken Schwaber, Mike Beedle, Jeff Sutherland and others have contributed significantly to the evolution of Scrum over the last decade. Scrum has garnered increasing popularity in the agile software development community due to its simplicity, proven productivity, and ability to act as a wrapper for various engineering practices promoted by other agile methodologies.

With Scrum methodology, the "Product Owner" works closely with the team to identify and prioritize system functionality in form of a "Product Backlog". The Product Backlog consists of features, bug fixes, non-functional requirements, etc. – whatever needs to be done in order to successfully deliver a working software system. With priorities driven by the Product Owner, cross-functional teams estimate and sign-up to deliver "potentially shippable increments" of software during successive Sprints, typically lasting 30 days. Once a Sprint's Product Backlog is committed, no additional functionality can be added to the Sprint except by the team. Once a Sprint has been delivered, the Product Backlog is analyzed and reprioritized, if necessary, and the next set of functionality is selected for the next Sprint.

Scrum methodology has been proven to scale to multiple teams across very large organizations with 800+ people. See how VersionOne supports Scrum Sprint Planning by making it easier to manage your Product Backlog.

Lean and Kanban Software Development

Lean Software Development is an iterative agile methodology originally developed by Mary and Tom Poppendieck. Lean Software Development owes much of its principles and practices to the Lean Enterprise movement, and the practices of companies like Toyota. Lean Software Development focuses the team on delivering Value to the customer, and on the efficiency of the "Value Stream," the mechanisms that deliver that Value. The main principles of Lean methodology include:

- Eliminating Waste
- Amplifying Learning

- Deciding as Late as Possible
- Delivering as Fast as Possible
- Empowering the Team
- Building Integrity In
- Seeing the Whole

Lean methodology eliminates waste through such practices as selecting only the truly valuable features for a system, prioritizing those selected, and delivering them in small batches. It emphasizes the speed and efficiency of development workflow, and relies on rapid and reliable feedback between programmers and customers. Lean uses the idea of work product being "pulled" via customer request. It focuses decision-making authority and ability on individuals and small teams, since research shows this to be faster and more efficient than hierarchical flow of control. Lean also concentrates on the efficiency of the use of team resources, trying to ensure that everyone is productive as much of the time as possible. It concentrates on concurrent work and the fewest possible intra-team workflow dependencies. Lean also strongly recommends that automated unit tests be written at the same time the code is written.

The Kanban Method is used by organizations to manage the creation of products with an emphasis on continual delivery while not overburdening the development team. Like Scrum, Kanban is a process designed to help teams work together more effectively.

Kanban is based on 3 basic principles:

- **Visualize what you do today (workflow):** seeing all the items in context of each other can be very informative
- **Limit the amount of work in progress (WIP):** this helps balance the flow-based approach so teams don 't start and commit to too much work at once
- **Enhance flow:** when something is finished, the next highest thing from the backlog is pulled into play
- Kanban promotes continuous collaboration and encourages active, ongoing learning and improving by defining the best possible team workflow. See how VersionOne supports Kanban software development.

Crystal

The **Crystal methodology is one of the most lightweight, adaptable approaches to software development**. Crystal is actually comprised of a family of agile methodologies such as Crystal Clear, Crystal Yellow, Crystal Orange and others, whose unique characteristics are driven by several factors such as team size, system criticality, and project priorities. This Crystal family addresses the realization that each project may require a slightly tailored set of policies, practices, and processes in order to meet the project 's unique characteristics.

Several of the key tenets of Crystal include teamwork, communication, and simplicity, as well as reflection to frequently adjust and improve the process. Like other agile process methodologies, Crystal promotes early, frequent delivery of working software, high user involvement, adaptability, and the removal of bureaucracy or distractions. Alistair Cockburn, the originator of Crystal, has released a book, *Crystal Clear: A Human-Powered Methodology for Small Teams*.

Dynamic Systems Development Method (DSDM)

DSDM, dating back to 1994, grew out of the need to provide an industry standard project delivery framework for what was referred to as Rapid Application Development (RAD) at the time.

While RAD was extremely popular in the early 1990 ‘s, the RAD approach to software delivery evolved in a fairly unstructured manner. As a result, the DSDM Consortium was created and convened in 1994 with the goal of devising and promoting a common industry framework for rapid software delivery.

Since 1994, the DSDM methodology has evolved and matured to provide a comprehensive foundation for planning, managing, executing, and scaling agile process and iterative software development projects.

DSDM is based on nine key principles that primarily revolve around business needs/value, active user involvement, empowered teams, frequent delivery, integrated testing, and stakeholder collaboration.

DSDM specifically calls out “fitness for business purpose” as the primary criteria for delivery and acceptance of a system, focusing on the useful 80% of the system that can be deployed in 20% of the time.

Requirements are baselined at a high level early in the project. Rework is built into the process, and all development changes must be reversible.

Requirements are planned and delivered in short, fixed-length time-boxes, also referred to as iterations, and requirements for DSDM projects are prioritized using MoSCoW Rules:

M *– Must have requirements*
S *– Should have if at all possible*
C *– Could have but not critical*
W *– Won ‘t have this time, but potentially later*

All critical work must be completed in a DSDM project. It is also important that not every requirement in a project or time-box is considered critical.

Within each time-box, less critical items are included so that if necessary, they can be removed to keep from impacting higher priority requirements on the schedule.

The DSDM project framework is independent of, and can be implemented in conjunction with, other iterative methodologies such as Extreme Programming and the Rational Unified Process.

Feature-Driven Development (FDD)

The FDD variant of agile methodology was originally developed and articulated by Jeff De Luca, with contributions by M.A. Rajashima, Lim Bak Wee, Paul Szego, Jon Kern and Stephen Palmer. The first incarnations of FDD occurred as a result of collaboration between De Luca and OOD thought leader Peter Coad. FDD is a model-driven, short-iteration process. It begins with establishing an overall model shape. Then it continues with a series of two-week “design by feature, build by feature” iterations. The features are small, “useful in the eyes of the client” results. FDD designs the rest of the development process around feature delivery using the following eight practices:

- Domain Object Modeling
- Developing by Feature
- Component/Class Ownership
- Feature Teams

- Inspections
- Configuration Management
- Regular Builds
- Visibility of progress and results

FDD recommends specific programmer practices such as “Regular Builds” and “Component/Class Ownership”.

FDD’s proponents claim that it scales more straightforwardly than other approaches, and is better suited to larger teams. Unlike other agile methods, FDD describes specific, very short phases of work, which are to be accomplished separately per feature. These include Domain Walkthrough, Design, Design Inspection, Code, Code Inspection, and Promote to Build.

The notion of “Domain Object Modeling” is increasingly interesting outside the FDD community, following the success of Eric Evans’ book *Domain-Driven Design*.

Extreme Programming (XP)

XP, originally described by Kent Beck, has emerged as one of the most popular and controversial agile methodologies. XP is a disciplined approach to delivering high-quality software quickly and continuously. It promotes high customer involvement, rapid feedback loops, continuous testing, continuous planning, and close teamwork to deliver working software at very frequent intervals, typically every 1-3 weeks.
The original XP recipe is based on four simple values □“ simplicity, communication, feedback, and courage □“ and twelve supporting practices:

- Planning Game
- Small Releases
- Customer Acceptance Tests
- Simple Design
- Pair Programming
- Test-Driven Development
- Refactoring
- Continuous Integration
- Collective Code Ownership
- Coding Standards
- Metaphor
- Sustainable Pace

Don Wells has depicted the XP process in a popular diagram.

In XP, the “Customer” works very closely with the development team to define and prioritize granular units of functionality referred to as “User Stories”. The development team estimates, plans, and delivers the highest priority user stories in the form of working, tested software on an iteration-by-iteration basis. In order to maximize productivity, the practices provide a supportive, lightweight framework to guide a team and ensure high-quality software.

- Concepts, Terms, Understanding of Agile Development

XP has its own vocabulary – Refactoring, Technical Debt, Timeboxing, Last Responsible Moment, Stories, Iterations, Velocity, Theory of Constraints, Mindfulness, etc.

- Refactoring
- **Technical Debt** – avoid shortcuts, keep design simple, refactor relentlessly
- ***Timeboxing*** meetings, research can reduce wasted discussion
- **Stories** represent self-contained, individual elements of the project. Stories are customer-centric, describing results in terms of business results.
- **Iteration** is a full cycle of design-code-verify-release practiced by XP teams. It’s a time-box that is usually one to three weeks long.
- **Velocity** is a simple way of mapping estimates to the calendar, It’s the total of estimates for the stories finished in an iteration
- **Theory of Constraints** – every system has a single constraint that determines the overall throughput of the system. Let’s assume that programmers are the constraint. Then allow programmers to set pace and their estimates are used for planning.
- **Mindfulness** – or in other words **AGILITY** – the ability to respond to change. So pay attention to detail and change.
-

- Agile Methods and choosing one like XP (Extreme Programming) method
- XP Lifecycle, how it works – Planning, Analysis, Design and Coding, Testing, Deployment

- The XP Team, the Whole Team and On-Site Customers, The Product Manager, Domain experts (Subject Experts), Interactional Designers, Business Analysts, Programmers (Developers), Designers and architects, Technical Specialists, Testers, Coaches, the Project Manager and the Project Community (the ecosystem within which the team exists), stakeholders, executive sponsor, etc. Note the exact structure of your team isn't important as long as it has all the knowledge it needs. The makeup of your team will depend more on your organization's traditions than anything else. You don't need to hire one person per role since some people can fill multiple roles. Someone MUST perform those duties even if no has a specific job title saying so. Suggest having one person as Product Manager and one as Programmer-Coach and other roles may blend. One idea is to use both programmers and customers to do the testers jobs.

- In new team, team size can be 4 to 10 programmers, an optimal team size is 6 programmers (one acting as coach) and 4 customers, 1 tester, 1 PMP, that is 12 people in total. A 20-person team is an advanced XP team. All team members should sit with team full-time and give the project their complete attention. Avoid fractional assignment as seen in matrix-managed organizations.
-

Adopting XP

- XP's applicability is based on organizations and people and not types of projects
- Prerequisites

  (1) Management Support, (2) Team Agreement and (3) A Co-Located Team
  (4) On-Site Customers, (5) Right Team Size
  (6) Use all the Practices
  - & Recommendations
    - A Brand-New Codebase and Strong Design Skills
    - A language that's easy to refactor
    - A friendly cohesive Team
    - Go!
  - XP for a distributed team – is what I need to learn and quick!

Links or References

- https://www.versionone.com/agile-101/agile-methodologies/
- https://www.codeproject.com/Articles/604417/Agile-software-development-methodologies-and-how-t
- https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm
- https://www.guru99.com/agile-scrum-extreme-testing.html
- https://www.pivotaltracker.com/
-

Session 3 Design Patterns in Android

- MVC
- MVP
- MVVM
- Singleton, use of a single object, single instance with a class
- Consumer-Producer with threads
-

Session 4 Reactive Programming, Rx, RxJava, RxAndroid

In computing, **reactive programming** is an asynchronous programming paradigm concerned with data streams and the propagation of change. This means that it becomes possible to express static (e.g. arrays) or dynamic (e.g. event emitters) *data streams* with ease via the employed programming language(s), and that an inferred dependency within the associated *execution model* exists, which facilitates the automatic propagation of the change involved with data flow.

"Reactive programming is programming with asynchronous data streams."

Angular 2.0 uses RxJS with is a **functional reactive programming** (FRP) library **Reactive programming** is not **functional programming**. **Reactive programming** is **programming** with asynchronous data streams (observables). ... The angular.io docs are also full of examples of internal class state and imperative state mutations.

- https://www.slideshare.net/LukaJacobowitz/reactive-programming-in-the-browser-feat-scalajs-and-rx
- https://www.slideshare.net/kasun04/reactive-programming-in-java-8-with-rxjava
- https://www.meetup.com/Thessaloniki-NET-Meetup/events/237158391/?_cookie-check=XdVrHZi_KNEhdjdQ
- http://slideplayer.com/slide/2558108/
-
- http://reactivex.io/
- https://gist.github.com/staltz/868e7e9bc2a7b8c1f754
- https://en.wikipedia.org/wiki/Reactive_programming
- https://medium.com/@tkssharma/reactive-programming-rx-js-introduction-32bf963eee1b
- http://www.introtorx.com/content/v1.0.10621.0/01_WhyRx.html
- http://www.introtorx.com/content/v1.0.10621.0/01_WhyRx.html
- https://www.slideshare.net/kasun04/reactive-programming-in-java-8-with-rxjava
- https://www.slideshare.net/InfoQ/reactive-programming-with-rx
- https://www.infoq.com/presentations/rx-service-architecture
-
- http://www.vogella.com/tutorials/RxJava/article.html
- https://www.infoq.com/articles/rxjava-by-example
- http://blog.danlew.net/2014/09/15/grokking-rxjava-part-1/
-

Notes on Rx

When is Rx appropriate?

Rx offers a natural paradigm for dealing with sequences of events. A sequence can contain zero or more events. Rx proves to be most valuable when composing sequences of events.

Should use Rx

Managing events like these is what Rx was built for:

- UI events like mouse move, button click
- Domain events like property changed, collection updated, "Order Filled", "Registration accepted" etc.
- Infrastructure events like from file watcher, system and WMI events
- Integration events like a broadcast from a message bus or a push event from WebSockets API or other low latency middleware like Nirvana
- Integration with a CEP engine like StreamInsight or StreamBase.

Interestingly Microsoft's CEP product StreamInsight, which is part of the SQL Server family, also uses LINQ to build queries over streaming events of data.
Rx is also very well suited for introducing and managing concurrency for the purpose of *offloading*. That is, performing a given set of work concurrently to free up the current thread. A very popular use of this is maintaining a responsive UI.

You should consider using Rx if you have an existing *IEnumerable<T>* that is attempting to model data in motion. While *IEnumerable<T>can* model data in motion (by using lazy evaluation like yield return), it probably won't scale. Iterating over an *IEnumerable<T>* will consume/block a thread. You should either favor the non-blocking nature of Rx via either IObservable<T> or consider the async features in .NET 4.5.

Could use Rx
Rx can also be used for asynchronous calls. These are effectively sequences of one event.

- Result of a Task or Task<T>
- Result of an APM method call like *FileStream* BeginRead/EndRead
-

You may find the using TPL, Dataflow or async keyword (.NET 4.5) proves to be a more natural way of composing asynchronous methods. While Rx can definitely help with these scenarios, if there are other more appropriate frameworks at your disposal you should consider them first.

Rx can be used, but is less suited for, introducing and managing concurrency for the purposes of *scaling* or performing *parallel* computations. Other dedicated frameworks like TPL (Task Parallel Library) or C++ AMP are more appropriate for performing parallel compute intensive work.

See more on TPL, Dataflow, async and C++ AMP at Microsoft's Concurrency homepage.

Won't use Rx

Rx and specifically *IObservable<T>* is not a replacement for *IEnumerable<T>*. I would not recommend trying to take something that is naturally pull based and force it to be push based.

- Translating existing *IEnumerable<T>* values to *IObservable<T>* just so that the code base can be "more Rx"
- Message queues. Queues like in MSMQ or a JMS implementation generally have transactionality and are by definition sequential. I feel *IEnumerable<T>* is a natural fit for here.

By choosing the best tool for the job your code should be easier to maintain, provide better performance and you will probably get better support.

Rx in action
Adopting and learning Rx can be an iterative approach where you can slowly apply it to your infrastructure and domain. In a short time, you should be able to have the skills to produce code, or reduce existing code, to queries composed of simple operators. For example, this simple ViewModel is all I needed to code to integrate a search that is to be executed as a user types.

```
public class MemberSearchViewModel: INotifyPropertyChanged
{
        //Fields removed...
        public MemberSearchViewModel (IMemberSearchModel memberSearchModel,
        ISchedulerProvider schedulerProvider)
        {
                _memberSearchModel = memberSearchModel;
                //Run search when SearchText property changes
                this.PropertyChanges (vm => vm.SearchText)
                .Subscribe (Search);
        }
```

```
        //Assume INotifyPropertyChanged implementations of properties...
        public string SearchText { get; set; }
        public bool IsSearching { get; set; }
        public string Error { get; set; }
        public ObservableCollection<string> Results { get; }

 //Search on background thread and return result on dispatcher.
 private void Search(string searchText)
 {
        using (_currentSearch) { }
        IsSearching = true;
        Results.Clear();
        Error = null;
        _currentSearch = _memberSearchModel.SearchMembers(searchText)
        .Timeout (TimeSpan.FromSeconds(2))
        .SubscribeOn (_schedulerProvider.TaskPool)
        .ObserveOn (_schedulerProvider.Dispatcher)
        .Subscribe (
        Results.Add,
        ex =>
        {
                IsSearching = false;
                Error = ex.Message;
        },
        () => { IsSearching = false; });
 }
...
}
```

While this code snippet is fairly small it supports the following requirements:

- Maintains a responsive UI
- Supports timeouts
- Knows when the search is complete
- Allows results to come back one at a time
- Handles errors
- Is unit testable, even with the concurrency concerns
- If a user changes the search, cancel current search and execute new search with new text.

To produce this sample is almost a case of composing the operators that match the requirements into a single query. The query is small, maintainable, declarative and far less code than "rolling your own". There is the added benefit of reusing a well-tested API. The less code *you* have to write, the less code *you* have to test, debug and maintain.

Creating other queries like the following is simple:

- calculating a moving average of a series of values e.g. **service level agreements** for average latencies or downtime
- combining event data from multiple sources e.g.: **search results** from Bing, Google and Yahoo, or **sensor data** from Accelerometer, Gyro, Magnetometer or temperatures
- grouping data e.g. **tweets** by topic or user, or **stock prices** by delta or liquidity
- filtering data e.g. **online game servers** within a region, for a specific game or with a minimum number of participants.

Push is here. Arming yourself with Rx is a powerful way to meet users' expectations of a push world. By understanding and composing the constituent parts of Rx you will be able to make short work of complexities of processing incoming events. Rx is set to become a day-to-day part of your coding experience.

RxJava Notes

In reactive programming the consumer reacts to the data as it comes in. This is the reason why asynchronous programming is also called reactive programming. Reactive programming allows to propagates event changes to registered observers.

Reactivex is a project which provides implementations for this concept for different programming languages. It describes itself as:

The Observer pattern done right. ReactiveX is a combination of the best ideas from the Observer pattern, the Iterator pattern, and functional programming.

RxJava is the Java implementation of this concept. RxJava is published under the Apache 2.0 license. RxJava provides Java API for asynchronous programming with observable streams.

2. Build blocks for RxJava

The build blocks for RxJava code are the following:

- **observables** representing sources of data
- **subscribers (or observers)** listening to the observables
- a set of methods for modifying and composing the data

An observable emits items; a subscriber consumes those items.

2.1. Observables

Observables are the sources for the data. Usually they start providing data once a subscriber starts listening. An observable may emit any number of items (including zero items). It can terminate either successfully or with an error. Sources may never terminate, for example, an observable for a button click can potentially produce an infinite stream of events.

2.2. Subscribers

A observable can have any number of subscribers. If a new item is emitted from the observable, the onNext() method is called on each subscriber. If the observable finishes its data flow successful, the onComplete() method is called on each subscriber. Similar, if the observable finishes its data flow with an error, the onError() method is called on each subscriber.

3. RxJava example

A very simple example written as JUnit4 test is the following:

```
package com.vogella.android.rxjava.simple;

import org.junit.Test;

import io.reactivex.Observable;
```

```
import static junit.framework.Assert.assertTrue;


public class RxJavaUnitTest {
   String result="";

   // Simple subscription to a fix value
   @Test
   public void returnAValue(){
      result = "";
      Observable<String> observer = Observable.just("Hello"); // provides datea
      observer.subscribe(s -> result=s); // Callable as subscriber
      assertTrue(result.equals("Hello"));
   }
}
```

RxJava Notes

3.1. Why doing asynchronous programming

Reactive programming provides a simple way of asynchronous programming. This allows to simplify the asynchronously processing of potential long running operations. It also provides a defined way of handling multiple events, errors and termination of the event stream. Reactive programming provides also a simplified way of running different tasks in different threads. For example, widgets in SWT and Android have to be updated from the UI thread and reactive programming provides ways to run observables and subscribers in different threads.

It is also possible to convert the stream before its received by the observers. And you can chain operations, e.g., if a API call depends on the call of another API Last but not least, reactive programming reduces the need for state variables, which can be the source of errors.

RxJava Notes

3.2. Adding RxJava 2 to a Java project

As of this writing the version 2.1.1 is currently the released one. Replace the version with your desired version.

To use RxJava in a Gradle build, add the following as dependency.

```
compile group: 'io.reactivex.rxjava2', name: 'rxjava', version: '2.1.1'
```

For Maven, you can add RxJava via the following snippet.

```
<dependency>
   <groupId>io.reactivex.rxjava2</groupId>
   <artifactId>rxjava</artifactId>
   <version>2.0.4</version>
</dependency>
```

For OSGi environments, e.g., Eclipse RCP development, https://dl.bintray.com/simon-scholz/RxJava-OSGi/ can be used as p2 update site.

4. Creating Observables, subscribing to them and disposing them

4.1. Creating observables

You can create different types of observables.

Table 1. Obervable types

Type	Description
Flowable<T>	Emits 0 or n items and terminates with an success or an error event. Supports backpressure, which allows to contro fast a source emits items.
Observable<T>	Emits 0 or n items and terminates with an success or an error event.
Single<T>	Emits either a single item or an error event. The reactive version of a method call.
Maybe<T>	Succeeds with an item, or no item, or errors. The reactive version of an Optional.
Completable	Either completes with an success or with an error event. It never emits items. The reactive version of a Runnable.

RxJava Notes

RxJava Notes

An example for the usage of Flowable, is when you process touch events. You cannot control the user who is doing these touch events, but you can tell the source to emit the events on a slower rate in case you cannot processes them at the rate the user produces them.

The following shows an example for the creation of an observable.

```
Observable<Todo> todoObservable = Observable.create(new ObservableOnSubscribe<Todo>() {
        @Override
        public void subscribe(ObservableEmitter<Todo> emitter) throws Exception {
            try {
```

```
                List<Todo> todos = RxJavaUnitTest.this.getTodos();
                for (Todo todo : todos) {
                    emitter.onNext(todo);
                }
                emitter.onComplete();
            } catch (Exception e) {
                emitter.onError(e);
            }
        }
    });
```

Using lambdas, the same statement can be expressed as:

```
Observable<Todo> todoObservable = Observable.create(emitter -> {
    try {
        List<Todo> todos = getTodos();
        for (Todo todo : todos) {
            emitter.onNext(todo);
        }
        emitter.onComplete();
    } catch (Exception e) {
        emitter.onError(e);
    }
});
```

The following is an example for a `Maybe`.

```
Maybe<List<Todo>> todoMaybe = Maybe.create(emitter -> {
    try {
        List<Todo> todos = getTodos();
        if(todos != null && !todos.isEmpty()) {
            emitter.onSuccess(todos);
        } else {
            emitter.onComplete();
        }
    } catch (Exception e) {
        emitter.onError(e);
    }
});
```

`java.util.Optional` has a value

`java.util.Optional` contains no value → null

An error occurred

4.2. Convenience methods to create observables

RxJava provides several convenience methods to create observables

- `Observable.just("Hello")` - Allows to create an observable as wrapper around other data types
- `Observable.fromIterable()` - takes an `java.lang.Iterable<T>` and emits their values in their order in the data structure

- Observable.fromArray() - takes an array and emits their values in their order in the data structure
- Observable.fromCallable() - Allows to create an observable for a java.util.concurrent.Callable<V>
- Observable.fromFuture() - Allows to create an observable for a java.util.concurrent.Future
- Observable.interval() - An observable that emits Long objects in a given interval

Similar methods exists for the other data types, e.g., *Flowable.just(), Maybe.just() and Single.just.

4.3. Subscribing in RxJava

To receive the data emitted from an observable you need to subscribe to it. observables offer a large variety of subscribe methods.

```
Observable<Todo> todoObservable = Observable.create(emitter -> { ... });

// Simply subscribe with a io.reactivex.functions.Consumer<T>, which will be informed onNext()
Disposable disposable = todoObservable.subscribe(t -> System.out.print(t));

// Dispose the subscription when not interested in the emitted data any more
disposable.dispose();

// Also handle the error case with a second io.reactivex.functions.Consumer<T>
Disposable subscribe = todoObservable.subscribe(t -> System.out.print(t), e -> e.printStackTrace());
```

There is also a subscribeWith method on observable instances, which can be used like this:

```
DisposableObserver<Todo> disposableObserver = todoObservable.subscribeWith(new
DisposableObserver<Todo>() {

    @Override
    public void onNext(Todo t) {
    }

    @Override
    public void onError(Throwable e) {
    }

    @Override
    public void onComplete() {
    }
});
```

4.4. Disposing subscriptions and using CompositeDisposable

When listers or subscribers are attached they usually are not supposed to listen eternally.

So it could happen that due to some state change the event being emitted by an observable might be not interesting any more.

```
import io.reactivex.Single;
import io.reactivex.disposables.Disposable;
import io.reactivex.observers.DisposableSingleObserver;

Single<List<Todo>> todosSingle = getTodos();
```

```
Disposable disposable = todosSingle.subscribeWith(new DisposableSingleObserver<List<Todo>>() {

    @Override
    public void onSuccess(List<Todo> todos) {
        // work with the resulting todos
    }

    @Override
    public void onError(Throwable e) {
        // handle the error case
    }
});

// continue working and dispose when value of the Single is not interesting any more
disposable.dispose();
```

> The Single class and other observable classes offer different subscribe methods, which return a Disposable object.

When working with multiple subscriptions, which may become obsolete due to the same state change using a CompositeDisposable is pretty handy to dispose a collection of subscriptions.

```
import io.reactivex.Single;
import io.reactivex.disposables.Disposable;
import io.reactivex.observers.DisposableSingleObserver;
import io.reactivex.disposables.CompositeDisposable;

CompositeDisposable compositeDisposable = new CompositeDisposable();

Single<List<Todo>> todosSingle = getTodos();

Single<Happiness> happiness = getHappiness();

compositeDisposable.add(todosSingle.subscribeWith(new DisposableSingleObserver<List<Todo>>() {

    @Override
    public void onSuccess(List<Todo> todos) {
        // work with the resulting todos
    }

    @Override
    public void onError(Throwable e) {
        // handle the error case
    }
}));

compositeDisposable.add(happiness.subscribeWith(new DisposableSingleObserver<Happiness>() {

    @Override
    public void onSuccess(Happiness happiness) {
        // celebrate the happiness :-D
    }

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

```
        System.err.println("Don't worry, be happy! :-P");
    }
}));

// continue working and dispose all subscriptions when the values from the Single objects are not interesting any more
compositeDisposable.dispose();
```

5. Caching values of completed observables

When working with observables doing async calls on every subscription on an observable is often not necessary.

It likely happens that observables are passed around in the application, without the need to do an such an expensive call all the time a subscription is added.

The following code does the expensive web query 4 times, even though doing this once would be fine, since the same `Todo` objects should be shown, but only in different ways.

```
Single<List<Todo>> todosSingle = Single.create(emitter -> {
    Thread thread = new Thread(() -> {
        try {
            List<Todo> todosFromWeb = // query a webservice

            System.out.println("Called 4 times!");

            emitter.onSuccess(todosFromWeb);
        } catch (Exception e) {
            emitter.onError(e);
        }
    });
    thread.start();
});

todosSingle.subscribe(... " Show todos times in a bar chart " ...);

showTodosInATable(todosSingle);

todosSingle.subscribe(... " Show todos in gant diagram " ...);

anotherMethodThatsSupposedToSubscribeTheSameSingle(todosSingle);
```

The next code snippet makes use of the `cache` method, so that the `Single` instance keeps its result, once it was successful for the first time.

```
Single<List<Todo>> todosSingle = Single.create(emitter -> {
    Thread thread = new Thread(() -> {
        try {
            List<Todo> todosFromWeb = // query a webservice

            System.out.println("I am only called once!");

            emitter.onSuccess(todosFromWeb);
        } catch (Exception e) {
            emitter.onError(e);
        }
    });
```

```
    thread.start();
});

// cache the result of the single, so that the web query is only done once
Single<List<Todo>> cachedSingle = todosSingle.cache();

cachedSingle.subscribe(... " Show todos times in a bar chart " ...);

showTodosInATable(cachedSingle);

cachedSingle.subscribe(... " Show todos in gant diagram " ...);

anotherMethodThatsSupposedToSubscribeTheSameSingle(cachedSingle);
```

6. Conversion between types

It is easy to convert between different RxJava types.

Table 2. Conversion between types

From / To	Flowable	Observable	Maybe	Single	Comple
Flowable		toObservable()	reduce() elementAt() firstElement() lastElement() singleElement()	scan() elementAt() first()/firstOrError() last()/lastOrError() single()/singleOrError() all()/any()/count() (and more…)	ignoreEle
Observable	toFlowable()		reduce() elementAt() firstElement() lastElement() singleElement()	scan() elementAt() first()/firstOrError() last()/lastOrError() single()/singleOrError() all()/any()/count() (and more…)	ignoreEle
Maybe	toFlowable()	toObservable()		toSingle() sequenceEqual()	toComple
Single	toFlowable()	toObservable()	toMaybe()		toComple
Completable	toFlowable()	toObservable()	toMaybe()	toSingle() toSingleDefault()	

7. RxAndroid

7.1. Using RxAndroid

RxAndroid is an extension to RxJava. It providers a scheduler to run code in the main thread of Android. It also provides the ability to create a scheduler that runs on a Android handler class. With this schedulers, you can define an

observable which does its work in a background thread, and post our results to the main thread. This allows for example to replace a AsyncTaskimplementations which RxJava.

To use RxJava in Android add the following dependency to your build.gradle file.

```
compile 'io.reactivex.rxjava2:rxandroid:2.0.1'
compile 'io.reactivex.rxjava2:rxjava:2.0.8'
```

For example you can define a long running operation via the following observable.

```
final Observable<Integer> serverDownloadObservable = Observable.create(emitter -> {
        SystemClock.sleep(1000); // simulate delay
        emitter.onNext(5);
        emitter.onComplete();
    });
```

You can now subscribe to this observable. This triggers its execution and provide the subscribe with the required information.

For example, lets assume you assign this to a button.

```
serverDownloadObservable.
                        observeOn(AndroidSchedulers.mainThread()).
                        subscribeOn(Schedulers.io()).
                        subscribe(integer -> {
                            updateTheUserInterface(integer); // this methods updates the ui
                            view.setEnabled(true); // enables it again
                        });
            }
.subscribeOn(Schedulers.io())
.observeOn(AndroidSchedulers.mainThread());
```

The subscriber observes in the main thread

Observable is called outside the main thread

As we are only interested in the final result, we could also use a Single.

```
Subscription subscription = Single.create(new Single.OnSubscribe() {
        @Override
        public void call(SingleSubscriber singleSubscriber) {
            String result = doSomeLongRunningStuff();
            singleSubscriber.onSuccess(value);
        }
    })
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(new Action1() {
        @Override
        public void call(String value) {
            // onSuccess
            updateTheUserInterface(); // this methods updates the ui
        }
```

```
    }, new Action1() {
        @Override
        public void call(Throwable throwable) {
            // handle onError
        }
    });
```

7.2. Unsubscribe to avoid memory leaks

Observable.subscribe() returns a Subscription (if you are using a Flowable) or a Disposable object. To prevent a possible (temporary) memory leak, unsubscribe from your observables in the`onStop()` method of the activity or fragment. For example, for a Disposable object you could do the following:

```
@Override
  protected void onDestroy() {
      super.onDestroy();
      if (bookSubscription != null && !bookSubscription.isDisposed()) {
          bookSubscription.dispose();
      }
  }
```

8. Exercise: First steps with RxJava and RxAndroid

Create a new project with the com.vogella.android.rxjava.simple top level package name.

8.1. Gradle dependencies

Add the following dependencies to your app/build.gradle file.

```
compile 'com.android.support:recyclerview-v7:23.1.1'
compile 'io.reactivex.rxjava2:rxandroid:2.0.1'
compile 'io.reactivex.rxjava2:rxjava:2.0.8'
compile 'com.squareup.okhttp:okhttp:2.5.0'
testCompile 'junit:junit:4.12'
```

Also enable the usage of Java 8 in your app/build.gradle file.

```
android {
  // more stuff
   compileOptions {
      sourceCompatibility JavaVersion.VERSION_1_8
      targetCompatibility JavaVersion.VERSION_1_8
   }
}
```

8.2. Create activities

Change your main layout file to the following.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical"
   >

   <Button
```

```
      android:id="@+id/first"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:onClick="onClick"
      android:text="First"
      />

   <Button
      android:id="@+id/second"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:onClick="onClick"
      android:text="Second"

      />

   <Button
      android:id="@+id/third"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:onClick="onClick"
      android:text="Third"

      />
</LinearLayout>
```

Create three activities:

- RxJavaSimpleActivity
- BooksActivity
- ColorsActivity

Create the activity_rxjavasimple.xml layout file.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   >

   <Button
      android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Server"
      />
   <Button
      android:id="@+id/toastbutton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Toast"
      android:onClick="onClick"
```

```
    />

  <TextView
    android:id="@+id/resultView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Result"
    />
</LinearLayout>
```

activity_colors.xml

In RxJavaSimpleActivity create a observable which simulates a long running operation (10 secs) and afterwards returns the number 5. Subscribe to it via a button click, disable the button

```
package com.vogella.android.rxjava.simple;

import android.os.Bundle;
import android.os.SystemClock;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.RecyclerView;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

import io.reactivex.Observable;
import io.reactivex.android.schedulers.AndroidSchedulers;
import io.reactivex.disposables.CompositeDisposable;
import io.reactivex.disposables.Disposable;
import io.reactivex.schedulers.Schedulers;


public class RxJavaSimpleActivity extends AppCompatActivity {

  RecyclerView colorListView;
  SimpleStringAdapter simpleStringAdapter;
  CompositeDisposable disposable = new CompositeDisposable();
  public int value =0;

  final Observable<Integer> serverDownloadObservable = Observable.create(emitter -> {
    SystemClock.sleep(10000); // simulate delay
    emitter.onNext(5);
    emitter.onComplete();
  });

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_rxjavasimple);
    View view = findViewById(R.id.button);
    view.setOnClickListener(v -> {
      v.setEnabled(false); // disables the button until execution has finished
      Disposable subscribe = serverDownloadObservable.
          observeOn(AndroidSchedulers.mainThread()).
          subscribeOn(Schedulers.io()).
```

```
                subscribe(integer -> {
                    updateTheUserInterface(integer); // this methods updates the ui
                    v.setEnabled(true); // enables it again
                });
            disposable.add(subscribe);
        });
    }

    private void updateTheUserInterface(int integer) {
        TextView view = (TextView) findViewById(R.id.resultView);
        view.setText(String.valueOf(integer));
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (disposable!=null && !disposable.isDisposed()) {
            disposable.dispose();
        }
    }

    public void onClick(View view) {
        Toast.makeText(this, "Still active " + value++, Toast.LENGTH_SHORT).show();
    }
}
```

Create an adapter for a recycler view.

```
package com.vogella.android.rxjava.simple;

import android.content.Context;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.List;

/**
 * Adapter used to map a String to a text view.
 */
public class SimpleStringAdapter extends RecyclerView.Adapter<SimpleStringAdapter.ViewHolder> {

    private final Context mContext;
    private final List<String> mStrings = new ArrayList<>();

    public SimpleStringAdapter(Context context) {
        mContext = context;
    }

    public void setStrings(List<String> newStrings) {
        mStrings.clear();
```

```
        mStrings.addAll(newStrings);
        notifyDataSetChanged();
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.string_list_item, parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, final int position) {
        holder.colorTextView.setText(mStrings.get(position));
        holder.itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(mContext, mStrings.get(position), Toast.LENGTH_SHORT).show();
            }
        });
    }

    @Override
    public int getItemCount() {
        return mStrings.size();
    }

    public static class ViewHolder extends RecyclerView.ViewHolder {

        public final TextView colorTextView;

        public ViewHolder(View view) {
            super(view);
            colorTextView = (TextView) view.findViewById(R.id.color_display);
        }
    }
}
```

Implement `ColorsActivity` which uses a observable to receive a list of colors.

Create the activity_colors.xml layout file.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >
    <android.support.v7.widget.RecyclerView
        android:id="@+id/color_list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        />
</FrameLayout>
package com.vogella.android.rxjava.simple;
```

```
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;

import java.util.ArrayList;
import java.util.List;

import io.reactivex.Observable;
import io.reactivex.disposables.Disposable;


public class ColorsActivity extends AppCompatActivity {

    RecyclerView colorListView;
    SimpleStringAdapter simpleStringAdapter;
    private Disposable disposable;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        configureLayout();
        createObservable();
    }

    private void createObservable() {
        Observable<List<String>> listObservable = Observable.just(getColorList());
        disposable = listObservable.subscribe(colors -> simpleStringAdapter.setStrings(colors));

    }

    private void configureLayout() {
        setContentView(R.layout.activity_colors);
        colorListView = (RecyclerView) findViewById(R.id.color_list);
        colorListView.setLayoutManager(new LinearLayoutManager(this));
        simpleStringAdapter = new SimpleStringAdapter(this);
        colorListView.setAdapter(simpleStringAdapter);
    }

    private static List<String> getColorList() {
        ArrayList<String> colors = new ArrayList<>();
        colors.add("red");
        colors.add("green");
        colors.add("blue");
        colors.add("pink");
        colors.add("brown");
        return colors;
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (disposable!=null && !disposable.isDisposed()) {
            disposable.dispose();
        }
    }
```

```
}
```

Create the following (fake) server implementation.

```
package com.vogella.android.rxjava.simple;

import android.content.Context;
import android.os.SystemClock;

import java.util.ArrayList;
import java.util.List;

/**
 * This is a fake REST client.
 *
 * It simulates making blocking calls to an REST endpoint.
 */
public class RestClient {
   private Context mContext;

   public RestClient(Context context) {
      mContext = context;
   }

   public List<String> getFavoriteBooks() {
      SystemClock.sleep(8000);// "Simulate" the delay of network.
      return createBooks();
   }

   public List<String> getFavoriteBooksWithException() {
      SystemClock.sleep(8000);// "Simulate" the delay of network.
      throw new RuntimeException("Failed to load");
   }

   private List<String> createBooks() {
      List<String> books = new ArrayList<>();
      books.add("Lord of the Rings");
      books.add("The dark elf");
      books.add("Eclipse Introduction");
      books.add("History book");
      books.add("Der kleine Prinz");
      books.add("7 habits of highly effective people");
      books.add("Other book 1");
      books.add("Other book 2");
      books.add("Other book 3");
      books.add("Other book 4");
      books.add("Other book 5");
      books.add("Other book 6");
      return books;
   }
}
```

Create the activity_books.xml layout file.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<FrameLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   >

   <ProgressBar
      android:id="@+id/loader"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center"
      />

   <android.support.v7.widget.RecyclerView
      android:id="@+id/books_list"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:visibility="gone"
      />

</FrameLayout>
```

Also implement the BooksActivity activity.

```
package com.vogella.android.rxjava.simple;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.View;
import android.widget.ProgressBar;

import java.util.List;

import io.reactivex.Observable;
import io.reactivex.android.schedulers.AndroidSchedulers;
import io.reactivex.disposables.Disposable;
import io.reactivex.schedulers.Schedulers;


public class BooksActivity extends AppCompatActivity {

   private Disposable bookSubscription;
   private RecyclerView booksRecyclerView;
   private ProgressBar progressBar;
   private SimpleStringAdapter stringAdapter;
   private RestClient restClient;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      restClient = new RestClient(this);
      configureLayout();
      createObservable();
```

```
    }

    private void createObservable() {
        Observable<List<String>> booksObservable =
                Observable.fromCallable(() -> restClient.getFavoriteBooks());
        bookSubscription = booksObservable.
                subscribeOn(Schedulers.io()).
                observeOn(AndroidSchedulers.mainThread()).
                subscribe(strings -> displayBooks(strings));
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (bookSubscription != null && !bookSubscription.isDisposed()) {
            bookSubscription.dispose();
        }
    }

    private void displayBooks(List<String> books) {
        stringAdapter.setStrings(books);
        progressBar.setVisibility(View.GONE);
        booksRecyclerView.setVisibility(View.VISIBLE);
    }

    private void configureLayout() {
        setContentView(R.layout.activity_books);
        progressBar = (ProgressBar) findViewById(R.id.loader);
        booksRecyclerView = (RecyclerView) findViewById(R.id.books_list);
        booksRecyclerView.setLayoutManager(new LinearLayoutManager(this));
        stringAdapter = new SimpleStringAdapter(this);
        booksRecyclerView.setAdapter(stringAdapter);
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (bookSubscription!=null && !bookSubscription.isDisposed()) {
            bookSubscription.dispose();
        }
    }
}
```

8.3. Implement a long running implementation via a Callable

A java.util.Callable is like a runnable but it can throw an exception and return a value.

The following activity implement a observable created based on a Callable. During the subscription a progressbar will be make visible and once the process finishes the progressbar is hidden again and a text view is updated.

The long running operation will run in the background, the update of the UI will happen in the main thread.

Here is the activity_scheduler.xml layout file:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
```

```
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >

    <Button
        android:id="@+id/scheduleLongRunningOperation"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:lines="3"
        android:text="Start something long"
        android:layout_marginStart="12dp"
        android:layout_alignParentStart="true"
        />

    <TextView
        android:id="@+id/messagearea"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentStart="true"
        android:text=""
        android:layout_below="@+id/scheduleLongRunningOperation"
        />

    <ProgressBar
        android:id="@+id/progressBar"
        style="?android:attr/progressBarStyle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="gone"
        android:layout_alignBottom="@+id/scheduleLongRunningOperation"
        android:layout_toEndOf="@+id/scheduleLongRunningOperation"
        />

</RelativeLayout>
```

[[source, java]

```
package com.vogella.android.rxjava.simple;

import android.os.Bundle;
import android.os.SystemClock;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.ProgressBar;
import android.widget.TextView;

import java.util.concurrent.Callable;

import io.reactivex.Observable;
import io.reactivex.android.schedulers.AndroidSchedulers;
import io.reactivex.disposables.Disposable;
import io.reactivex.observers.DisposableObserver;
import io.reactivex.schedulers.Schedulers;
```

```
/** Demonstrates a long running operation of the main thread
 * during which a  progressbar is shown
 *
 */
public class SchedulerActivity extends AppCompatActivity {

    private Disposable subscription;
    private ProgressBar progressBar;
    private TextView messagearea;
    private View button;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        configureLayout();
        createObservable();
    }

    private void createObservable() {
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (subscription != null && !subscription.isDisposed()) {
            subscription.dispose();
        }
    }

    private void configureLayout() {
        setContentView(R.layout.activity_scheduler);
        progressBar = (ProgressBar) findViewById(R.id.progressBar);
        messagearea = (TextView) findViewById(R.id.messagearea);
        button  = findViewById(R.id.scheduleLongRunningOperation);
        button.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
//                progressBar.setVisibility(View.VISIBLE);
                Observable.fromCallable(callable).
                        subscribeOn(Schedulers.io()).observeOn(AndroidSchedulers.mainThread()).
                        doOnSubscribe(disposable ->
                                {
                                    progressBar.setVisibility(View.VISIBLE);
                                    button.setEnabled(false);
                                    messagearea.setText(messagearea.getText().toString() +"\n" +"Progressbar set visible" );
                                }
                        ).
                        subscribe(getDisposableObserver());
            }
        });
    }

    Callable<String> callable = new Callable<String>() {
        @Override
        public String call() throws Exception {
```

```
            return doSomethingLong();
        }
    };

    public String doSomethingLong(){
        SystemClock.sleep(1000);
        return "Hello";
    }

    /**
     * Observer
     * Handles the stream of data:
     */
    private DisposableObserver<String> getDisposableObserver() {
        return new DisposableObserver<String>() {

            @Override
            public void onComplete() {
                messagearea.setText(messagearea.getText().toString() +"\n" +"OnComplete" );
                progressBar.setVisibility(View.INVISIBLE);
                button.setEnabled(true);
                messagearea.setText(messagearea.getText().toString() +"\n" +"Hidding Progressbar" );
            }

            @Override
            public void onError(Throwable e) {
                messagearea.setText(messagearea.getText().toString() +"\n" +"OnError" );
                progressBar.setVisibility(View.INVISIBLE);
                button.setEnabled(true);
                messagearea.setText(messagearea.getText().toString() +"\n" +"Hidding Progressbar" );
            }

            @Override
            public void onNext(String message) {
                messagearea.setText(messagearea.getText().toString() +"\n" +"onNext " + message );
            }
        };
    }
}
```

9. Testing RxJava Observables and Subscriptions

9.1. Testing the observables

`Flowable` can be tested with `io.reactivex.subscribers.TestSubscriber`. Non-backpressured Observable, Single, Maybe and Completable can be tested with `io.reactivex.observers.TestObserver`.

```
@Test
    public void anObservableStreamOfEventsAndDataShouldEmitsEachItemInOrder() {

        Observable<String> pipelineOfData = Observable.just("Foo", "Bar");

        pipelineOfData.subscribe(testObserver);

        List<Object> dataEmitted = testObserver.values();
        assertThat(dataEmitted).hasSize(2);
        assertThat(dataEmitted).containsOnlyOnce("Foo");
```

```
        assertThat(dataEmitted).containsOnlyOnce("Bar");
    }
```

All base reactive types now have a test() method. This is a huge convenience for returning TestSubscriber or TestObserver.

```
TestSubscriber<Integer> ts = Flowable.range(1, 5).test();

TestObserver<Integer> to = Observable.range(1, 5).test();

TestObserver<Integer> tso = Single.just(1).test();

TestObserver<Integer> tmo = Maybe.just(1).test();

TestObserver<Integer> tco = Completable.complete().test();
```

10. Exercise: Writing unit tests for RxJava

10.1. Write small unit test

Create a small test to use RxJava in a test.

```
package com.vogella.android.rxjava.simple;

import org.junit.Test;

import java.util.List;

import io.reactivex.Observable;
import io.reactivex.ObservableEmitter;
import io.reactivex.ObservableOnSubscribe;
import io.reactivex.observers.TestObserver;

import static junit.framework.Assert.assertTrue;


public class RxJavaUnitTest {
    String result="";

    // Simple subscription to a fix value
    @Test
    public void returnAValue(){
        result = "";
        Observable<String> observer = Observable.just("Hello"); // provides datea
        observer.subscribe(s -> result=s); // Callable as subscriber
        assertTrue(result.equals("Hello"));
    }

    @Test public void test(){
        Observable<Todo> todoObservable = Observable.create(new ObservableOnSubscribe<Todo>() {
            @Override
            public void subscribe(ObservableEmitter<Todo> emitter) throws Exception {
                try {
                    List<Todo> todos = RxJavaUnitTest.this.getTodos();
                    if (todos!=null){
                        throw new NullPointerException("todos was null");
```

```
                }
                for (Todo todo : todos) {
                    emitter.onNext(todo);
                }
                emitter.onComplete();
            } catch (Exception e) {
                emitter.onError(e);
            }
        }
    });
    TestObserver<Object> testObserver = new TestObserver<>();
    todoObservable.subscribeWith(testObserver);
    testObserver.assertError(NullPointerException.class);

  }

  private List<Todo> getTodos() {
    return null;
  }

  public class Todo {
  }
}
```

The following code demonstrates the usage of `Callable` together with OkHttp and RxJava.

```
package com.vogella.android.rxjava.simple;

import org.junit.Test;

import java.util.List;

import io.reactivex.Observable;
import io.reactivex.ObservableEmitter;
import io.reactivex.ObservableOnSubscribe;
import io.reactivex.observers.TestObserver;

import static junit.framework.Assert.assertTrue;


public class RxJavaUnitTest {
  String result="";

  // Simple subscription to a fix value
  @Test
  public void returnAValue(){
    result = "";
    Observable<String> observer = Observable.just("Hello"); // provides datea
    observer.subscribe(s -> result=s); // Callable as subscriber
    assertTrue(result.equals("Hello"));
  }

  @Test public void test(){
    Observable<Todo> todoObservable = Observable.create(new ObservableOnSubscribe<Todo>() {
      @Override
```

```
        public void subscribe(ObservableEmitter<Todo> emitter) throws Exception {
            try {
                List<Todo> todos = RxJavaUnitTest.this.getTodos();
                if (todos!=null){
                    throw new NullPointerException("todos was null");
                }
                for (Todo todo : todos) {
                    emitter.onNext(todo);
                }
                emitter.onComplete();
            } catch (Exception e) {
                emitter.onError(e);
            }
        }
    });
    TestObserver<Object> testObserver = new TestObserver<>();
    todoObservable.subscribeWith(testObserver);
    testObserver.assertError(NullPointerException.class);

}

private List<Todo> getTodos() {
    return null;
}

public class Todo {
}
}
```

End for RxJava Notes

RxAndroid Notes

RxJava, at its core, is about two things: Observables and Observers. Observables are said to “emit” values. Their counterpart, Observers, watch Observables by “subscribing” to them.

Observers can take actions when an Observable emits a value, when the Observable says an error has occurred, or when the Observable says that it no longer has any values to emit.

All three of these actions are encapsulated in the Observer interface.
The corresponding functions are `onNext ( )`, *onError ( )*, and *onCompleted ( )*.

The **two main actors in RX world** are:

- Observable -- it emits events (in form of data)
- Observer -- it subscribes to Observable in order to receive its events

Many Observers can subscribe to single Observable, manipulate data streams using operators (more on this later on), which is the real strength of RX approach.

Functional programming allows us to greatly decouple dependencies and separate business logic, that operates within clearly defined boundaries of simple streams of data objects.

Although ReactiveX has very well-defined concepts and the API is mature, at first, everyone struggles a bit with more advanced concepts. It's not easy to shift your mind from classic imperative programming to functional programming, as well as intuitively pick the right RX operators from the vast array of possible solutions. Live examples are far better than theory in the case of RX.

Familiarity with Java 8 Streams helps a lot, but all you need to understand basics of RX is here anyway.

Reactive programming is an extension of the Observer software design pattern, where an object has a list of Observers that are dependent on it, and these Observers are notified by the object whenever it’s state changes.

There are two basic and very important items in reactive programming, Observables and Observers.

Observables publish values, while Observers subscribe to Observables, watching them and reacting when an Observable publishes a value.

In simpler terms:

- An Observable performs some action, and publishes the result.
- An Observer waits and watches the Observable, and reacts whenever the Observable publishes results.

There are three different changes that can occur on an Observable that the Observer reacts to. These are:

1. Publishing a value
2. Throwing an error
3. Completed publishing all values

A class that implements the Observer interface must provide methods for each of the three changes above:

1. An onNext () method that the Observable calls whenever it wishes to publish a new value
2. An onError () method that’s called exactly once, when an error occurs on the Observable.
3. An onCompleted () method that’s called exactly once, when the Observable completes execution.

So an Observable that has an Observer subscribed to it will call the Observer’s onNext () zero or more times, as long as it has values to publish, and terminates by either calling onError () or onCompleted ().

- https://code.tutsplus.com/tutorials/getting-started-with-reactivex-on-android--cms-24387
- http://www.androidauthority.com/reactive-programming-with-rxandroid-711104/

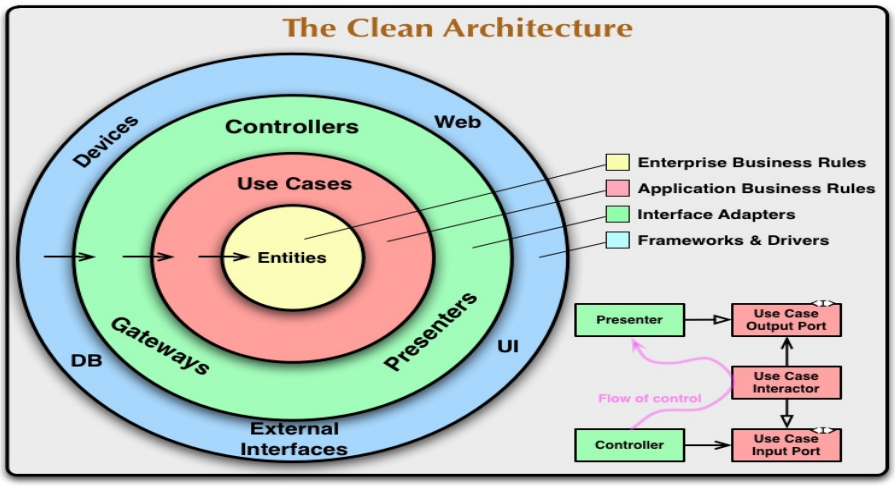
- https://medium.com/@kurtisnusbaum/rxandroid-basics-part-1-c0d5edcf6850
- https://medium.com/crunching-rxandroid
- https://x-team.com/blog/introduction-reactivex-android/

End of Rx, RFP, RxJava, RxAndroid

Day 5 Session 4 Clean Architecture and Android

How Clean Architecture works

The key rule behind Clean Architecture is: **The Dependency Rule**. The gist of this is simply that dependencies are encapsulated in each "ring" of the architecture model and these dependencies can only point inward.



Clean Architecture keeps details like web frameworks and databases in the outer layers while important business rules and policies are housed in the inner circles and have no knowledge of anything outside of themselves. Considering this, you can start to see how it achieves a very *clean* separation of concerns. By ensuring your business rules and core domain logic in the inner circles are completely devoid of any external dependencies or 3rd party libraries means they must be expressed using pure C# POCO classes which makes testing them much easier.

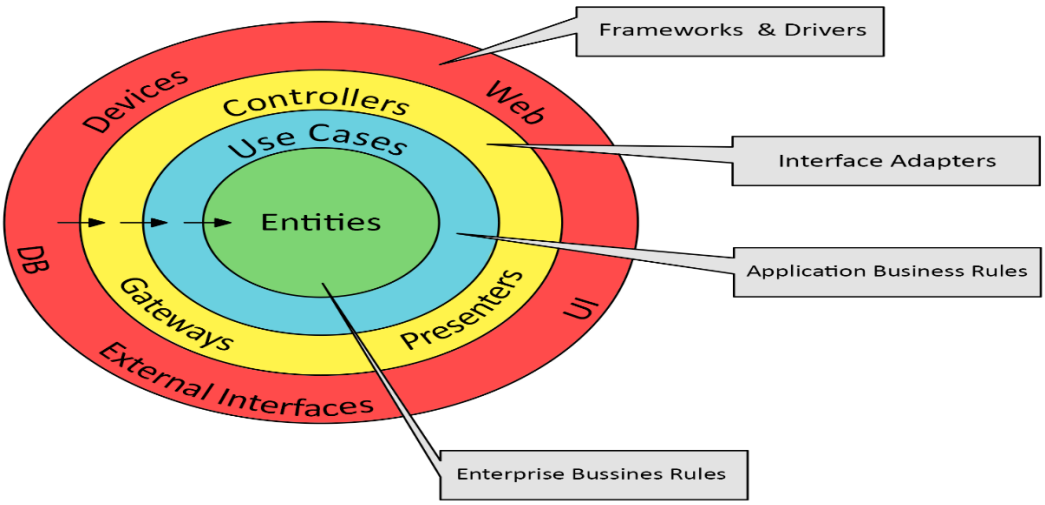
What Is Clean Architecture?

Clean Architecture builds upon the previously introduced four concepts and aligns the project with best practices like the *Dependency Inversion Principle* or *Use Cases*. It also aims for a maximum independence of any frameworks or tools that might stay in the way of application's testability or their replacement.

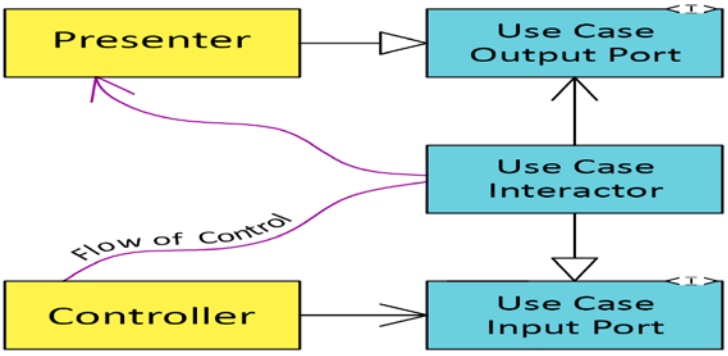
Clean Architecture divides our system into four layers, usually represented by circles:

- **Entities,** which contain enterprise-wide business rules. You can think of them as about Domain Entities a la DDD.
- **Use cases,** which contain application-specific business rules. These would be counterparts to *Application Services* with the caveat that each class should focus on one particular *Use Case*.
- **Interface adapters,** which contain adapters to peripheral technologies. Here, you can expect MVC, Gateway implementations and the like.
- **Frameworks and drivers**, which contain tools like databases or framework. By default, you don't code too much in this layer, but it's important to clearly state the place and priority that those tools have in your architecture.

Between the circles, there is a strong *dependency rule* – no code in the inner circle can directly reference a piece of code from the outer circle. All outward communication should happen via interfaces. It's exactly the same *dependency rule* as we introduced in the Onion Architecture post.



Apart from the layers, *Clean Architecture* gives us some tips about the classes we need to implement. As you can see in the picture below, the flow of control from the *Controller* to the *Use Case* goes through an *Input Port* interface and flows to the *Presenter* through an *Output Port* interface. This ensures that the *Use Case* and the user interface are properly decoupled. We'll see an example of this later in the implementation section.



The Essence of Clean Architecture

I see two things that make *Clean Architecture* distinct and potentially more effective than other architectural styles: strong adherence to the *Dependency Inversion Principle* and *Use Case* orientation.

Strong Adherence to DIP

Similar to its Onion cousin, *Clean Architecture* introduces the *Dependency Inversion Principle* at the architectural level. This way, it explicitly states the priorities between different kinds of objects in your system. In a way, *Clean Architecture* does a better job at this, as it leaves no doubt about the tools like frameworks or databases – they have a dedicated layer outside all others.

Use Case Orientation

Similar to what we've seen in *Package by Feature*, *Clean Architecture* promotes vertical slicing of the code and leaving the layers mostly at the class level. The major difference between the two is that instead of focusing on a blurry concept of a *feature*, it reorients the packaging towards *Use Cases*. This is important as, ultimately, in any application that has some sort of GUI, one could identify real *Use Cases*. It's also important to note that entities sit

in a different layer as in complex systems, one *Use Case* can orchestrate several entities to cooperate and categorizing it by the type of the entity would be artificial.

Implementing Clean Architecture

We can't be 100% sure about how Uncle Bob would implement a Clean Architecture today, as his book about it comes out in July (I preordered it already and will do a review or rehash of this post then), but we can look at the GitHub repository of his Clean Code Case study: example - https://github.com/cleancoders/CleanCodeCaseStudy/tree/master/src/cleancoderscom

Packaging

As you can see, the *Entities* and *Use Cases* layers have their own separate packages, while the other layers can be identified only conceptually. The socketserver.http, and view packages can be considered a part of the *Frameworks and Drivers*, while gateways package is a little bit ambiguous – their implementation is surely the *Interface Adapters* layer, but their interfaces conceptually belong to *Use Cases*. My guess is that the interfaces are extracted to a separate package so they can be shared between different *Use Cases*. But it's just a guess!

By looking at the usecases.codecastSummeries, we can get more insight into how a complete *Use Case*package looks like. As you can see, it accommodates all classes related to the execution of a particular *Use Case*: the view, controller, presenter, boundaries, view and response models, and the *Use Case* class itself. This might be a lot more classes than you usually see in your projects when you execute an *Application Service*, but that's what it takes to go perfectly *Clean*.

Internals

If you dug deeper into the implementation of the project's classes, you'd see no annotation there other than @Override. That's because of the frameworks being at the very outer layer of the architecture – the code is not allowed to reference them directly. You might ask, *how could I leverage Spring in such a project?* Well, if you really wanted to, you'd have to do some XML configurations or do it using @Configuration and @Bean classes. No @Service, no @Autowired, sorry!

My Extra Advice

Pulling off a *Clean Architecture* might be a demanding task, especially if you worked with a *Package by Layer*, fat controllers kind of project before. And even if you get the idea and necessary skills to implement it, your colleagues might not. They might want to do this anyway, but simply forget to add interfaces or work around framework annotations when necessary. One way to prevent some of these issues could be to create a Maven module for each of the layers so that breaking a rule won't even compile. At the same time, if you don't have these already, introducing Pair Programming or Code Reviews will help you to prevent people from messing up with the dependency declarations (circular dependencies would not work, but adding Spring both to the *Use Cases* and *Adapters* module would!).

Benefits of a Clean Architecture

- **Screaming** – *Use Cases* are clearly visible in the project's structure
- **Flexible** – you should be able to switch frameworks, databases or application servers like pairs of gloves
- **Testable** – all the interfaces around let you setup the test scope and outside interactions in any way you want
- **Could play well with best practices like DDD** – to be honest, I haven't seen it so far, but I also don't see anything stopping you from making an effective mix of DDD's Strategic and Tactical Patterns with *Clean Architecture*

Drawbacks of Clean Architecture

- **No Idiomatic Framework Usage** – the *dependency rule* is relentless in this area

- **Learning Curve** – it's harder to grasp than the other styles, especially considering the point above
- **Indirect** – there will be a lot more interfaces than one might expect (I don't see it as necessarily bad, but I've seen people pointing this out)
- **Heavy** – in the sense that you might end up with a lot more classes than you currently have in your projects (again, the extra classes are not necessarily bad)

When to Use Clean Architecture

Before I say something, let me note that I haven't tried implementing it in a professional context yet, so all of it is a gut feeling. We will probably get some more knowledgeable advice from Uncle Bob himself in his upcoming book.

If we consider *Clean Architecture*'s biggest drawbacks and its essence, I would derive the following criteria to consider:

- **Is the team skilled and/or convinced enough?** One might consider this lame, but if people just don't get it or they don't want to do this, imposing the rigor of *Clean Architecture* on them might be counter-productive.
- **Will the system outlive major framework releases?** Since we're talking about heavy technology flexibility here, it's important to consider if we'll ever capitalize on this benefit. My experience so far suggests that most systems will, even if the developers won't be in the company by then.
- **Will the system outlive the developers and stakeholder's employment?** Since *Clean Architecture* is so sound and makes *Use Cases* so clearly visible, systems that follow its principles will be much simpler to comprehend in the code, even if those who wrote it and asked for it are already gone.

Summary

Clean Architecture looks like a very carefully thought and effective architecture. It makes the big leap of recognizing the mismatch between *Use Cases* and *Entities* and puts the former in the driving seat of our system. It also gives a clear place for *Frameworks and Drivers* in our system – a separate layer outside all other layers. This, combined with the *dependency rule* might give us a plethora of benefits, but also might be way harder to pull off. In the end, it boils down to the question whether the system will live long enough so that the investment returns.

Links

- http://wiseassblog.com/android/software%20architecture/2017/06/21/clean-architecture-designing-use-case-or-interactors/
- https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html
- https://www.safaribooksonline.com/library/view/clean-architecture-a/9780134494272/
- https://android.jlelse.eu/a-complete-idiots-guide-to-clean-architecture-2422f428946f
- https://dzone.com/articles/remembering-clean-architecture
- http://five.agency/android-architecture-part-3-applying-clean-architecture-android/
- https://medium.com/@dmilicic/a-detailed-guide-on-developing-android-apps-using-the-clean-architecture-pattern-d38d71e94029
- https://www.entropywins.wtf/blog/2016/11/24/implementing-the-clean-architecture/
- http://luboganev.github.io/blog/clean-architecture-pt1/
-

End of Clean Architecture

Services and Messages

Defines a message containing a description and arbitrary data object that can be sent to a Handler. This object contains two extra int fields and an extra object field that allow you to not do allocations in many cases.

While the constructor of Message is public, the best way to get one of these is to call Message.obtain() or one of the Handler.obtainMessage()methods, which will pull them from a pool of recycled objects.

A `Service` is an application component that can perform long-running operations in the background, and it does not provide a user interface. Another application component can start a service, and it continues to run in the background even if the user switches to another application. Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC). For example, a service can handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

These are the three different types of services:

Foreground

A foreground service performs some operation that is noticeable to the user. For example, an audio app would use a foreground service to play an audio track. Foreground services must display a status bar icon. Foreground services continue running even when the user isn't interacting with the app.

Background

A background service performs an operation that isn't directly noticed by the user. For example, if an app used a service to compact its storage, that would usually be a background service.

Note: If your app targets API level 26 or higher, the system imposes restrictions on running background services when the app itself is not in the foreground. In most cases like this, your app should use a scheduled job instead.

Bound

A service is *bound* when an application component binds to it by calling `bindService()`. A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results, and even do so across processes with interprocess communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

Although this documentation generally discusses started and bound services separately, your service can work both ways—it can be started (to run indefinitely) and also allow binding. It's simply a matter of whether you implement a couple of callback methods: `onStartCommand()` to allow components to start it and `onBind()` to allow binding.

Regardless of whether your application is started, bound, or both, any application component can use the service (even from a separate application) in the same way that any component can use an activity—by starting it with an `Intent`. However, you can declare the service as *private* in the

manifest file and block access from other applications. This is discussed more in the section about Declaring the service in the manifest.

Caution: A service runs in the main thread of its hosting process; the service does **not** create its own thread and does **not** run in a separate process unless you specify otherwise. If your service is going to perform any CPU-intensive work or blocking operations, such as MP3 playback or networking, you should create a new thread within the service to complete that work. By using a separate thread, you can reduce the risk of Application Not Responding (ANR) errors, and the application's main thread can remain dedicated to user interaction with your activities.

The basics

Should you use a service or a thread?

A service is simply a component that can run in the background, even when the user is not interacting with your application, so you should create a service only if that is what you need.

If you must perform work outside of your main thread, but only while the user is interacting with your application, you should instead create a new thread. For example, if you want to play some music, but only while your activity is running, you might create a thread in `onCreate()`, start running it in `onStart()`, and stop it in `onStop()`. Also consider using `AsyncTask` or `HandlerThread` instead of the traditional `Thread` class. See the Processes and Threading document for more information about threads.

Remember that if you do use a service, it still runs in your application's main thread by default, so you should still create a new thread within the service if it performs intensive or blocking operations.
To create a service, you must create a subclass of `Service` or use one of its existing subclasses. In your implementation, you must override some callback methods that handle key aspects of the service lifecycle and provide a mechanism that allows the components to bind to the service, if appropriate. These are the most important callback methods that you should override:

`onStartCommand()`

The system invokes this method by calling `startService()` when another component (such as an activity) requests that the service be started. When this method executes, the service is started and can run in the background indefinitely. If you implement this, it is your responsibility to stop the service when its work is complete by calling `stopSelf()` or `stopService()`. If you only want to provide binding, you don't need to implement this method.

`onBind()`

The system invokes this method by calling `bindService()` when another component wants to bind with the service (such as to perform RPC). In your implementation of this method, you must provide an interface that clients use to communicate with the service by returning an `IBinder`. You must always implement this method; however, if you don't want to allow binding, you should return null.

`onCreate()`

The system invokes this method to perform one-time setup procedures when the service is initially created (before it calls either `onStartCommand()` or `onBind()`). If the service is already running, this method is not called.

`onDestroy()`

The system invokes this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, or receivers. This is the last call that the service receives.

If a component starts the service by calling `startService()` (which results in a call to `onStartCommand()`), the service continues to run until it stops itself with `stopSelf()` or another component stops it by calling `stopService()`.

If a component calls `bindService()` to create the service and `onStartCommand()` is *not* called, the service runs only as long as the component is bound to it. After the service is unbound from all of its clients, the system destroys it.

The Android system force-stops a service only when memory is low and it must recover system resources for the activity that has user focus. If the service is bound to an activity that has user focus, it's less likely to be killed; if the service is declared to run in the foreground, it's rarely killed. If the service is started and is long-running, the system lowers its position in the list of background tasks over time, and the service becomes highly susceptible to killing—if your service is started, you must design it to gracefully handle restarts by the system. If the system kills your service, it restarts it as soon as resources become available, but this also depends on the value that you return from `onStartCommand()`. For more information about when the system might destroy a service, see the Processes and Threading document.

In the following sections, you'll see how you can create the `startService()` and `bindService()` service methods, as well as how to use them from other application components.

Declaring a service in the manifest

You must declare all services in your application's manifest file, just as you do for activities and other components.

To declare your service, add a `<service>` element as a child of the `<application>` element. Here is an example:

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".ExampleService" />
      ...
  </application>
</manifest>
```

See the `<service>` element reference for more information about declaring your service in the manifest.

There are other attributes that you can include in the `<service>` element to define properties such as the permissions that are required to start the service and the process in which the service should run. The `android:name` attribute is the only required attribute—it specifies the class name of the service. After you publish your application, leave this name unchanged to avoid the risk of breaking code due to dependence on explicit intents to start or bind the service (read the blog post, Things That Cannot Change).

Caution: To ensure that your app is secure, always use an explicit intent when starting a `Service` and do not declare intent filters for your services. Using an implicit intent to start a service is a security hazard because you cannot be certain of the service that will respond to the intent, and the user cannot see which service starts. Beginning with Android 5.0 (API level 21), the system throws an exception if you call `bindService()` with an implicit intent.

Since the very beginning, Android's central focus has been the ability to multitask. In order to achieve it, Android takes a unique approach by allowing multiple applications to run at the same time. Applications are never explicitly closed by the user, but are instead left running at a low priority to be killed by the system when memory is low. This ability to keep processes waiting in the background speeds up app-switching later down the line.

https://developer.android.com/reference/android/os/Binder.html
http://www.androiddesignpatterns.com/2013/08/binders-death-recipients.html
https://developer.android.com/reference/android/os/IBinder.html

Developers learn early on that the key to how Android handles applications in this way is that **processes aren't shut down cleanly**. Android doesn't rely on applications being well-written and responsive to polite requests to exit. Rather, it brutally force-kills them without warning, allowing the kernel to immediately reclaim resources associated with the process. This helps prevent serious out of memory situations and gives Android total control over misbehaving apps that are negatively impacting the system.

For this reason, there is no guarantee that any user-space code (such as an Activity's `onDestroy()` method) will ever be executed when an application's process goes away.

Considering the limited memory available in mobile environments, this approach seems promising.

However, there is still one issue that needs to be addressed: *how should the system detect an application's death so that it can quickly clean up its state*? When an application dies, its state will be spread over dozens of system services (the Activity Manager, Window Manager, Power Manager, etc.) and several different processes.

These system services need to be notified immediately when an application dies so that they can clean up its state and maintain an accurate snapshot of the system. Enter death recipients.

Death Recipients

As it turns out, this task is made easy using the `Binder`'s "link-to-death" facility, which allows a process to get a callback when another process hosting a binder object goes away. In Android, any process can receive a notification when another process dies by taking the following steps:

1. First, the process creates a `DeathRecipient` callback object containing the code to be executed when the death notification arrives.
2. Next, it obtains a reference to a `Binder` object that lives in another process and calls its`linkToDeath(IBinder.DeathRecipient recipient, int flags)`, passing the `DeathRecipient`callback object as the first argument.

3. Finally, it waits for the process hosting the `Binder` object to die. When the Binder kernel driver detects that the process hosting the `Binder` is gone, it will notify the registered `DeathRecipient`callback object by calling its `binderDied()` method.

AIDL – Android Interface Definition Language

The actual AIDL mechanism should be a familiar one to Java developers: you provide an interface, and a tool (the *aidl* tool) will generate the necessary plumbing in order for other applications (clients) to communicate with your application (service) across process boundaries. Time for some concrete example.

Say we want to implement a **phone book service** so that other Android applications can do a look up by name and get a list of corresponding phone numbers. We start by creating a simple Interface to express that capability, by writing a *IPhoneBookService.aidl* file in our source directory:

```
package com.ts.phonebook.service;

/* PhoneBook remote service, provides a list of matching phone numbers given
a person's name*/
interface IPhoneBookService{

   List<String> lookUpPhone(String name);
}
```

- https://developer.android.com/guide/components/aidl.html
- https://dzone.com/articles/android-interface-definition
- https://stuff.mit.edu/afs/sipb/project/android/docs/guide/components/aidl.html
- https://www.survivingwithandroid.com/2014/03/android-remote-service-tutorial-aidl.html
- http://codetheory.in/android-interprocess-communication-ipc-with-aidl/
- http://techblogon.com/android-aidl-example-with-code-description-ipc/

Messages, message objects, message objects, message queues and messenger in android

Low-level class holding the list of messages to be dispatched by a `Looper`. Messages are not added directly to a MessageQueue, but rather through `Handler` objects associated with the Looper.

You can retrieve the MessageQueue for the current thread with `Looper.myQueue()`.

- https://developer.android.com/reference/android/os/MessageQueue.html
- https://developer.android.com/reference/android/os/Handler.html
- http://codetheory.in/android-handlers-runnables-loopers-messagequeue-handlerthread/
- https://stackoverflow.com/questions/12877944/what-is-the-relationship-between-looper-handler-and-messagequeue-in-android

Threads, threading, thread pools, thread communicates to another thread, thread management

- https://www.safaribooksonline.com/library/view/efficient-android-threading/9781449364120/ch04.html
- http://www.c-sharpcorner.com/interview-question/what-is-looper-handler-and-message-queue-in-android
- http://ww.vogella.com – Android Style Message Passing – communicating between threads using message queues

Wrapping APIs and third-party libraries

- https://softwareengineering.stackexchange.com/questions/107338/using-third-party-libraries-always-use-a-wrapper
- https://www.b4x.com/android/forum/threads/wrapping-android-library-projects.36410/
- https://github.com/johncarl81/parceler
- http://blog.mashape.com/how-to-use-mashapes-java-client-library/
- https://android.jlelse.eu/consuming-rest-api-using-retrofit-library-in-android-ed47aef01ecb
- http://www.androidauthority.com/how-to-hide-your-api-key-in-android-600583/
- http://www.androidauthority.com/use-remote-web-api-within-android-app-617869/
- https://developer.android.com/google/play/billing/billing_best_practices.html
- https://stackoverflow.com/questions/42557962/how-to-secure-remote-api-for-calls-not-coming-from-tyk
- https://stormpath.com/blog/build-user-authentication-for-android-app

Android UI Design Tool – Droid Draw

- http://www.authorcode.com/create-android-applcation-with-droiddraw/
- http://www.cas.mcmaster.ca/khedri/wp-content/uploads/COURSES/2014_3A04/SE3A04Lab9.pdf
- https://hackaday.com/2010/08/05/android-development-101-part-5droiddraw-information-tracker-completed/
- http://www.brighthub.com/mobile/google-android/articles/24494.aspx
- http://atutorialandroidstudio.blogspot.in/2015/07/android-studio-tutorial-create-layouts-with-DroidDraw.html
- https://www.slideshare.net/info_zybotech/how-to-create-ui-using-droid-draw
- http://guide.fluidui.com/tutorial/

Android Coding and Third-Party Libraries

- https://www.reddit.com/r/androiddev/comments/4nuo5o/what_exactly_does_the_timber_library_do/
- https://github.com/JakeWharton/timber
- https://www.originate.com/library/libraries

Networking in Android

- http://www.vogella.com/tutorials/AndroidNetworking/article.html
- https://www.raywenderlich.com/126770/android-networking-tutorial-getting-started
- https://www.tutorialspoint.com/android/android_network_connection.htm
- https://androidkennel.org/android-networking-tutorial-with-asynctask/
- https://developer.android.com/studio/profile/am-network.html

Networking and Third-Party Plugins

- https://solidgeargroup.com/android-priority-job-queue-background-tasks
  https://www.originate.com/library/libraries

Content Providers and Third-Party Plugins

- https://www.originate.com/library/libraries
- https://www.lynda.com/Android-tutorials/Overview-Cupboard/540500/612834-4.html
- https://code.neenbedankt.com/introducing-cupboard-simple-persistance-for-android/
- https://guides.codepath.com/android/Easier-SQL-with-Cupboard
- http://androidcustomviews.com/cupboard/

-

Content Provider – REALM Database and Android

- https://blog.realm.io/realm-for-android/?
- https://www.androidhive.info/2016/05/android-working-with-realm-database-replacing-sqlite-core-data/
- https://dzone.com/articles/realm-practical-use-in-android
- https://auth0.com/blog/integrating-realm-database-in-an-android-application/
- http://www.theappguruz.com/blog/realm-mobile-database-implementation-in-android
- https://www.thecrazyprogrammer.com/2016/12/android-realm-database-tutorial.html
- https://medium.com/the-android-guy/android-working-with-realm-database-replacing-sqlite-core-data-b95d981f74d1

```
Realm realm = Realm.getDefaultInstance();

// All writes are wrapped in a transaction
// to facilitate safe multi threading
realm.beginTransaction();

// Add a person
Person person = realm.createObject(Person.class);
person.setName("Young Person");
person.setAge(14);

realm.commitTransaction();

RealmResults<User> result = realm.where(User.class)
                                 .greaterThan("age", 10)  // implicit AND
                                 .beginGroup()
                                      .equalTo("name", "Peter")
                                      .or()
                                      .contains("name", "Jo")
                                 .endGroup()
                                 .findAll();
```

Memory Monitor Overview, Android Profiler and Memory Profiler

- https://developer.android.com/studio/profile/android-monitor.html
- https://developer.android.com/studio/profile/memory-profiler.html The Memory Profiler is a component in the Android Profiler that helps you identify memory leaks and memory churn that can lead to stutter, freezes, and even app crashes. It shows a real-time graph of your app's memory use, lets you capture a heap dump, force garbage collections, and track memory allocations.
- https://developer.android.com/studio/profile/investigate-ram.html
- https://developer.android.com/studio/profile/am-cpu.html CPU Monitor
- https://developer.android.com/studio/profile/am-network.html Network Monitor
- https://developer.android.com/studio/profile/gpu.html GPU Monitor
- https://developer.android.com/studio/profile/am-hprof.html The HPROF Viewer displays classes, instances of each class, and a reference tree to help you track memory usage and find memory leaks. HPROF is a binary heap dump format originally supported by J2SE.
- https://developer.android.com/studio/profile/am-allocation.html Android Monitor allows you to track memory allocation as it monitors memory use. Tracking memory allocation allows you to monitor where

objects are being allocated when you perform certain actions. Knowing these allocations enables you to adjust the method calls related to those actions to optimize app performance and memory use.

The Allocation Tracker does the following:

1. Shows when and where your code allocates object types, their size, allocating thread, and stack traces.
2. Helps recognize memory churn through recurring allocation/deallocation patterns.
3. Helps you track down memory leaks when used in combination with the HPROF Viewer. For example, if you see a bitmap object resident on the heap, you can find its allocation location with Allocation Tracker.

- https://developer.android.com/studio/profile/am-methodtrace.html Method Tracer
  It lets you view the call stack and timing information for your app. This information can help you optimize and debug your app

- https://developer.android.com/studio/profile/battery-historian.html

- https://developer.android.com/studio/profile/monitor.html

- https://developer.android.com/studio/profile/hierarchy-viewer.html Hierarchy Viewer is a tool built into Android Device Monitor that allows you to inspect the properties and layout speed for each view in your layout hierarchy. It can help you find performance bottlenecks caused by the structure of your view hierarchy, helping you then simplify the hierarchy and reduce overdraw.

Android Fragmentation Issue and Its Solutions from Google, or anyone else

The cause of Android fragmentation is not difficult to pinpoint. Such disparity in devices occurs simply because Android is an open-source operating system – in short, manufacturers are (within limits) allowed to use Android as they please, and are thus responsible for offering updates as they see fit. The problem here is obvious; not every manufacturer (or carrier, as we'll get to) will remain consistent with updates, and some Android versions running on devices may be so heavily modified that updates just don't make sense.

One big reason for fragmentation is manufacturers' insistence on "skinning" their versions of Android – that is, offering a unique take on Android customized for a particular phone. This is why phones running MIUI will look infinitely different than a Nexus device, although both are running the same operating system beneath the visual and functional differences.

- http://www.androidauthority.com/android-fragmentation-google-fix-it-713210/
- http://bgr.com/2017/05/12/android-8-0-update-fragmentation-solution/
- https://www.digitaltrends.com/mobile/what-is-android-fragmentation-and-can-google-ever-fix-it/
- http://www.techtimes.com/articles/185234/20161109/google-has-a-plan-to-solve-the-fragmentation-problem-of-android-is-android-extensions-the-answer.htm
- https://www.pastemagazine.com/articles/2015/01/what-google-is-doing-to-solve-the-android-fragment.html
- https://www.pastemagazine.com/articles/2017/05/can-google-solve-android-fragmentation-for-good.html
- https://www.theverge.com/2017/5/12/15632552/android-o-faster-updates-project-treble-google
-

Android and doing it the Clean Way with Clean Architecture

Generally, in Clean, code is separated into layers in an onion shape with one **dependency rule:** The inner layers should not know anything about the outer layers. Meaning that the **dependencies should point inwards**.

We work hard to create beautiful, scalable and maintainable Android apps. To achieve that goal, over the years we have tried different architecture approaches (MVP, MVVM, MVC and some custom architectures) until we finally encountered the **Clean Architecture**.

My example here is a Restaurant Booker app, which involves showing list of townships, getting the restaurants from selected townships and booking a table for it. So, I decided to take a feature of that app (getting a list of townships and showing it in an recycler view—Yep, a dead simple one)

To make the app to have three layer—Data, Domain, and Presentation.

- ***Data layer*** will include POJOs and means to get Data from cloud or local storage.
- ***Domain layer*** will include all business logic and interact between Data and Presentation layer by means of interface and interactors. The objective is to make the domain layer independent of anything, so the business logic can be tested without any dependency to external components.
- ***Presentation layer*** will include normal Activities and Fragments, which will only handle rendering views and will follow MVP pattern.

I added a fifth part the UI Module which I believe is the easiest part.

Dependency rule defines that concrete modules depend on the more abstract ones.
You might remember from the third part of this series that UI (*app*), DB – API (*data*) and Device (*device*) stuff is together in the outer ring. Meaning that they are on the same abstraction level. How do we connect them together then?

Ideally, these modules would depend only on the *domain* module.
But, we are dealing with Android here and things just cannot be perfect. Because we need to create our object graph and initialize things, modules sometimes depend on an another module other than the domain.
For example, we are creating object graph for dependency injection in the *app* module. That forces *app* module to know about all of the other modules.

Check out http://five.agency/android-architecture-part-4-applying-clean-architecture-on-android-hands-on/

Alternatively,

I can have a different set of things in my architecture – entities, use cases, repositories, presenters, device, DB & API, UI, Modules … see http://five.agency/android-architecture-part-3-applying-clean-architecture-android/

- https://medium.com/@dmilicic/a-detailed-guide-on-developing-android-apps-using-the-clean-architecture-pattern-d38d71e94029
- https://fernandocejas.com/2014/09/03/architecting-android-the-clean-way/
- http://five.agency/android-architecture-part-3-applying-clean-architecture-android/
- http://five.agency/android-architecture-part-4-applying-clean-architecture-on-android-hands-on/
- https://android.jlelse.eu/a-complete-idiots-guide-to-clean-architecture-2422f428946f
- http://luboganev.github.io/blog/clean-architecture-pt1/
- https://blog.uptech.team/clean-architecture-in-android-with-kotlin-rxjava-dagger-2-2fdc7441edfc
- https://blog.moove-it.com/the-clean-architecture-on-android/
-