Cheatography

Android Studio App Development Cheat Sheet Cheat Sheet
by thecontinentalreaper (thecontinentalreaper) via cheatography.com/171910/cs/361

Mobile App Development Lifecycle

Inception > **Design**(UX->UI) > **Development** > **Stabilization** > **Distribution**

IDEs for Mobile App Development

Android Studio (Java/Kotilin), XCode (Objective C/Swift, can only run on MacOS), Microsoft XNA (C#, for Windows Phone apps), cocos2d(multi-language and platform, for games), Unity (C#, cross-platform), React Native (JS, cross-platform), Flutter (Dart language), Apache Cordova (HTML5, CSS3 and JS, cross-platform), MiniProgram (for making lightweight apps that don't require installation and occupy little memory/space)

Design Patterns
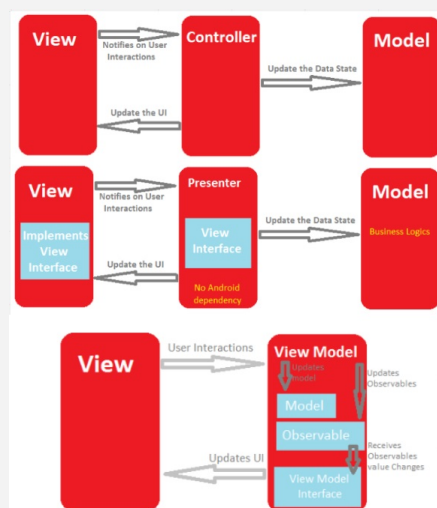
PATTERN	Dependency on Android APIS	XML Complexity	Unit Testability	Modular & SRP
MVC Controller	High	Low	Difficult	No
MVP Presenter	Low	Low	Good	Yes
MVVM ViewModel	Low or No dependency	Medium or High	Great	Yes

UI Design

Principles: User familiarity, consistency, minimal surprise, recoverability, user guidance, user diversity

UI Design Process
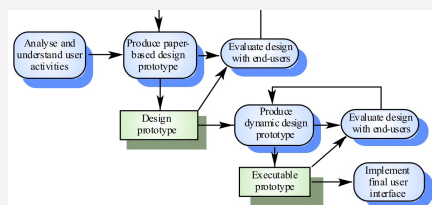
Broadcasts

Broadcasts are messages sent whenever an event of interests occurs from the Android System or from apps. Apps can register to receive certain broadcasts. They must be defined programatically in the code in addition to being declared in the manifest with intent filters. `<in ten t-f ilt - er> <action androi d:n ame - ="AC TIO N"/> </i nte nt- fil - ter>` BroadcastReceivers have an onReceive() function to check intent.action.

Server Connection

Must include the following permissions: `<us es- per mission androi d:n - ame ="an dro id.p er mis sio - n.I NTE RNE T" /> <us es- per - mission androi d:n ame ="an - dro id.p er mis sio n.A CCE - SS_ NET WOR K_S TAT E" />`
WebView is a View that can be display web pages in your app.

Java vs Kotilin

Functions	Variables
`public fun sum(a: Int, b: Int): Int { return a + b }`	`//val is immutable. var is mutable// val a: Int = 1 val b: String = "tim"{nl} //Types Int, String are optional and can be excluded//`

Nullable variable	Arrays
`var str S2? = " - hku " str = null`	`val num = arrayOf(1, 2, 3, 4) //implicit type declar ation val num = arrayO - f<I nt>(1, 2, 3, 4) //explicit type declaration`
Null safety `var str: S2? = " - hku " val I = S2?.length //if S2 is null, I is set o null`	`for (i in 0..num.si ze-1) { print( num[i]) }`

By **thecontinentalreaper** (thecontinentalreaper)

cheatography.com/thecontinentalreaper/

Not published yet.
Last updated 15th December, 2022.
Page 1 of 3.

Cheatography

Android Studio App Development Cheat Sheet Cheat Sheet
by thecontinentalreaper (thecontinentalreaper) via cheatography.com/171910/cs/361

Java vs Kotilin (cont)

Classes	UI Elements
`class Person(val name: String, val age: Int? = null) //Declares class and constr- uctor in one line! val person1 = Person- ("Sam", 20) //No new keyword`	`val edit_text1 = findVi ewB yId/ <Vi ew> (R.i d.e di tTextId) as EditText val button1 = findVi ewB yId <Vi ew> (R.i d.b ut ton1) as Button val listener = View.O nCl ick Lis tener {edit_text1.setText("Hello!")} button1.setOnClickListener(listener)`

Android Layout

Inflate the layout file in OnCreate():

```
public void onCrea te( Bundle
savedI nst anc eState) {
super.o nC rea te( sav edI nst-
anc eSt ate); setCon ten tVi-
ew( R.l ayo ut.a ct ivi ty_-
main);}
```
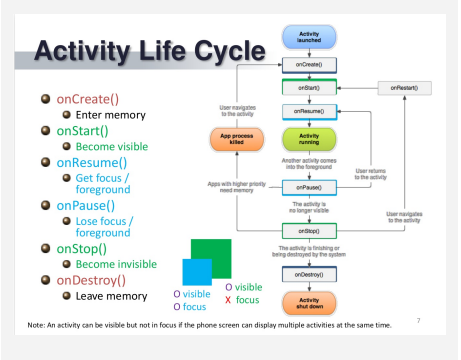
FrameLayout: Display a single item at a time. All elements are positioned based on top left of the screen, and multiple elements will overlap.

LinearLayout: Organizes elements along a single line, either horizontal or vertical (defined in XML property `androi d:o- rie nta tio n="h ori zon tal ")`

Android Layout (cont)

RelativeLayout: `androi d:l ayo ut_- aBo yTo/-bel ow/ toL eft Of/- toRa ightOf androi d:l ayo ut_ ali gnB ase- lin eVB ott om/ Lef t/R igh- t/Top`

TableLayout: `<Ta ble Lay out ><T- ableRo w><  Element androi d:l- ayo ut_col umn ="1"/ ></ Tab- leR ow>...< /T abl eLa you t>`

ConstraintLayout: Constrain a component to be in a position relative to another element. For example, `app:la you t_c ons- tra int Lef t_t oRi ght Of= "- @+i d/e lem ent " />` would constrain the left side of the target to the right side of the other element. Rember, it is **app:layout**, NOT **android:layout**.
Use sp for font size, as it scales with user font preferences, and dp for others as it changes based on different screen density.

Concurrency

A process is an instance of a program that is being executed or processed. They don't share resources. Switching between process is expensive. Threads are segments of processes and share memory. **Thread{...}.start()**
Main Thread is the UI thread which renders everything onscreen. Two rules of Android UI: **Do not block the UI thread. Do not access the UI toolkit from outside the UI thread.**
Runnable is a class that can be run inside a Thread with just 1 method: run().

Concurrency (cont)

Remember to use **myUIElement.post{ Runnable }** to force the Runnable object to join a queue so as to not break the rule. Handlers can also be used to update the UI thread (`handle r.post{ Runnable )`. Remember, **Threads cannot update UI, only Handlers**. Before running post() using Handler, call handler.removeCallbacks(run- nable) to remove any pending posts of the runnable in the queue so as to avoid repeated task. **handler.postDelayed(runn- able, time)** delays the runnable from starting until after the specified time..

Activity LifeCycle

Cheatography

Android Studio App Development Cheat Sheet Cheat Sheet
by thecontinentalreaper (thecontinentalreaper) via cheatography.com/171910/cs/361

Intents and Filters

Intents are messages sent between Activity, Service and Broad Receivers. **Explicit intents** are used within the application for tasks such as switching between activities. They specify which component should be called. Kotilin: `intent = Intent - (this, FooAct ivi ty: :cl - ass.java) startA cti vit y(i - ntent);` *Implicit intents* only specify the action to be performed, and are sent to the Android system which chooses which component should be used. Kotilin: `intent = Intent (In ten t.A CTI ON_ - VIEW, Uri.pa rse (UR LStr)); startA cti vit y(i ntent);` Activity needs **Intent Filter** to receive implicit intents. `<in ten t-f ilt er> <action androi d:name = " hku cs.m yi - nte ntf ilt er" /> <ca tegory androi d:name = " and roi d.i - nte nt.c at ego ry.D EF AUL T" /> </i nte nt- fil ter>` **Passing extra data**: Sender: `myInte nt.p ut - Ext ra( " ID", 6963)` Recpient: `val bundle: Bundle? = intent.extras bundle?.let {bundl e.apply { val inputS tring: String? = getStr - ing ("I D") // id = 6963}}`.

Fragment

Must implement: **onCreate()** - called when creating the fragment, should initialize essential components you want to retain when the fragment is paused or stopped, then resumed. **onCreateView** - called when the fragment draws its user interface for the first time. Optional: **onPause()** - called when the user is leaving the fragment.
Kotilin implementation

Fragment (cont)

```
val manager: Fragme ntM anager =
supportFragmentManager
val ft: Fragme ntT ran saction =
manager.beginTransaction()
if (fragment != null) ft.rep -
lac e(<ID of region in main
layout >, <class name of
fragme nt>(), <number tag to
represent fragme nt>); ft.com -
mit All owi ngS tat eLo ss();
```

Service

StartService(): runs indefinitely even if caller app dies. Simple, single task. No return result. Cannot be called back or modified after it is sent out.
BindService(): Can be bound to multiple components. Terminates if all callers die. Can be modified after being sent out.
A service can be both started and bound simulatenously.
Activating service: `val intent = Intent (this, HelloS erv - ice ::c las s.j ava); startS - erv ice (In tent);`
Return value of onStartCommand():
START_NOT_STICKY: Do not recreate after kill. Caller can restart unfinished jobs
START_STICKY: Recreate, but do not redeliver last intent. Continuous work but stateless, e.g. media players.
START_REDELIVER_INTENT: Recreate, and redeliver last intent. Actively performing a job, e.g. file download

Storage

SharedPreferences: Private primitive data in key-value pairs (persistent storage)
Bundles: Private primitive data in key-value pairs (temp storage for activity-fragment transfer)
Internal storage: Private data on device memory
External storage: Public data on internal or shared external storage, e.g., SD card
SQLite database: Structured data (table) in a private database