ANDROID DEVELOPMENT

PROGRAMMING LANGUAGE

- Developing Android native apps started with Java. Although it is still possible, Google more and more shifts to Kotlin (since 2019, Kotlin is the preferred language).
- Kotlin is a new programming language developed by JetBrains. It is compiled into Bytecode for the Java Virtual Machine.
- Kotlin and Java code can be mixed.

ANDROID DEVELOPMENT STUDIO

- Android Apps can be developed in different IDEs, most prominent is Android Studio.
- You can download the latest version of Android Studio here.
- Android Studio provides a wizard for generating a simple "Hello World" project.
- During the wizard, you need to provide a "Minimum SDK". The Minimum SDK defines the lowest API level your app will run on (i.e. API 21: Android 5.0 (Lollipop)).

EXERCISE

Open Android Studio and create a new project. In the wizard, use API 28 or 29 (whatever you have installed) as minimum SDK and "Basic Activity". Run the project.

VERSIONING

Android versions are managed using so called Api levels.
New os releases often provide new API levels.

Version	Released	API Level	Name
12	October 2021	31	Android 12
11	September 2020	30	Android 11
10	September 2019	29	Q
9.0	August 2018	28	Pie

↓ more ↓

Version	Released	API Level	Name
8.1	December 2017	27	Oreo
8.0	August 2017	26	Oreo
7.1	October 2016	25	Nougat
7.0	August 2016	24	Nougat

↓ more ↓

Version	Released	API Level	Name
6.0	October 2015	23	Marshmallow
5.1	March 2015	22	Lollipop
5.0	Nov 2014	21	Lollipop
4.4W	June 2014	20	Kitkat Watch
4.4	Oct 2013	19	Kitkat

↓ more ↓

Version	Released	API Level	Name
4.3	July 2013	18	Jelly Bean
4.2-4.2.2	Nov 2012	17	Jelly Bean
4.1-4.1.1	June 2012	16	Jelly Bean
4.0.3-4.0.4	Dec 2011	15	Ice Cream Sandwich
4.0-4.0.2	Oct 2011	14	Ice Cream Sandwich

Version	Released	API Level	Name
3.2	June 2011	13	Honeycomb
3.1.x	May 2011	12	Honeycomb
3.0.x	Feb 2011	11	Honeycomb

↓ more ↓

Version	Released	API Level	Name
2.3.3-2.3.4	Feb 2011	10	Gingerbread
2.3-2.3.2	Nov 2010	9	Gingerbread
2.2.x	June 2010	8	Froyo
2.1.x	Jan 2010	7	Eclair
2.0.1	Dec 2009	6	Eclair
2.0	Nov 2009	5	Eclair

↓ more ↓

Version	Released	API Level	Name
1.6	Sep 2009	4	Donut
1.5	May 2009	3	Cupcake
1.1	Feb 2009	2	Base
1.0	Oct 2008	1	Base

PLATFORM VERSIONS I

Android defines three different versions:

- minSdkVersion: The minimum api level your app will run on, contains the value you defined during the project wizard.
- compileSdkVersion: The api level that was used to build your app. Check Tools->Android->SDK Manager to see your installed SDKs.
- targetSdkVersion: The api level that was explicitly tested and works with this app. This is used by Android to optimize compatability code. This is checked at run time.

PLATFORM VERSIONS II

- Calling an Api larger than your minSdkVersion will raise a compile error.
- Android provides a number of support libraries that allow to use new features and still support older API levels (the support libraries have been moved to AndroidX, the concept remains the same).
- Check http://developer.android.com/tools/support-library/index.html for an overview.
- Check https://developer.android.com/jetpack/androidx/migrate/class-mappings for a list of mappings from support library to AndroidX.

PLATFORM VERSIONS III

The platform versions supported by an app are defined in build.gradle:

```
android {
    compileSdkVersion 22
    buildToolsVersion "22.0.1"

    defaultConfig {
        applicationId "ch.zhaw.testapplication"
        minSdkVersion 21
        targetSdkVersion 22
        versionCode 1
        versionName "1.0"
    }
```

ANDROID FILES

- AndroidManifest.xml: Defines the name of the app as well as the main entry point, the icons, style and many other things.
- res/: Subfolder that contains all resources (layouts, images, strings, ...).
- build.gradle: Build script for the Android builder tool gradle.
- gradle.properties, settings.gradle: Some files to store project settings.

ANDROID PACKAGE

An Android app is always packaged in an apk file (similar to iOS ipa). It is also a zip with a defined structure.

MATERIAL DESIGN

MATERIAL DESIGN

- Created for Android 5.0
- New widgets
- A new z property that represents the elevation of a view
- Shadows for elevation
- New animations
- Vector drawables

MATERIAL YOU

- New Design Language introduced in Android 12
- Supports numerous accessibility features
- Better support for different screen sizes
- Supports adaptable colors (ie dark mode)
- New Components

MATERIAL DESIGN VIDEOS

Google has published a number of videos for Android Material Design:

- Introduction to Material Design (1.55m)
- Material Design in the Google IO App (8.41m)
- Android 12: Designed for you (0:45m)

COLORS I

Material Design provides a color palette that you can use to define your colors. Basically, define three colors:

- colorPrimary: This is the main color of your app and used for the toolbar.
- colorPrimaryDark: This is the second color of your app, it should be a slightly darker color than colorPrimary. It is used for different components like the statusbar.
- colorAccent: The accent color is used to draw buttons, switches and sliders.

COLORS II

Additionally, there are colors for text, secondary text, disabled components and dividers. These colors are predefined by your theme (light or dark).

ICONS

Android provides further resources like icons that you can use for free:

https://material.io/icons/

DESIGN COMPONENTS

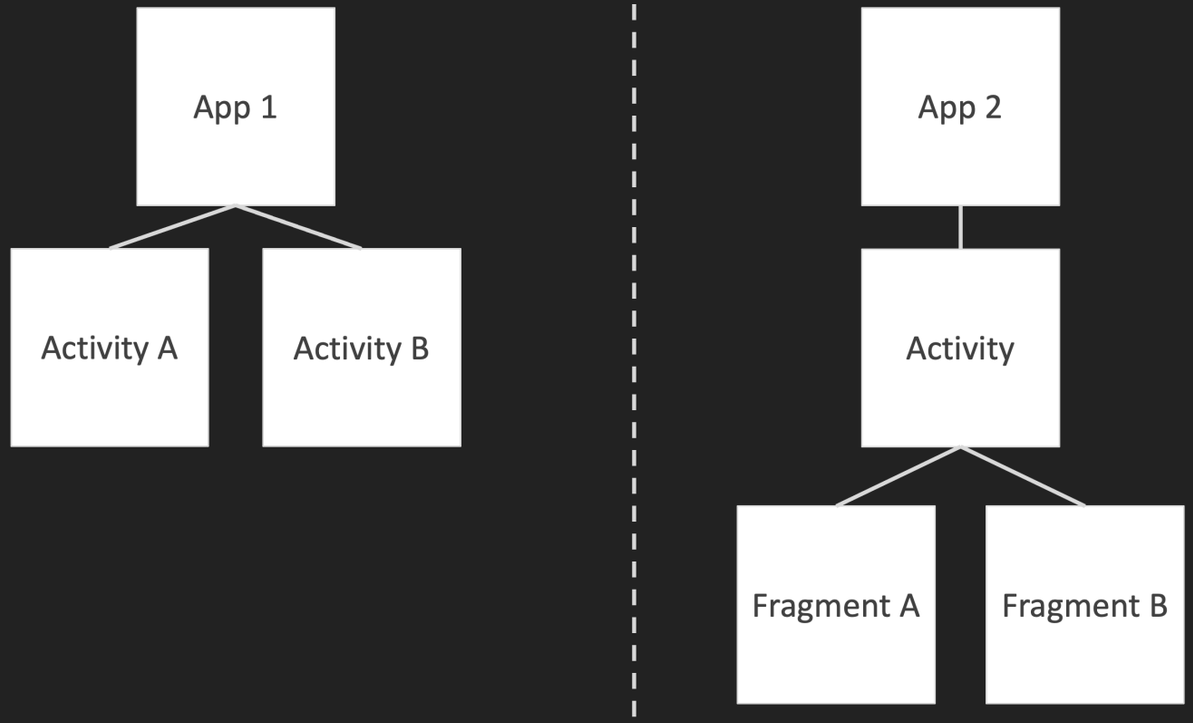
You can find a description of Material Design and of all components here. Components include:

- Cards: Display content for a single object.
- Chips: List elements that can be checked.
- Date Pickers: For selecting dates and ranges of dates.
- Snackbars: Provide feedback on an operation.

ACTIVITY AND FRAGMENTS

- Each "page" in the app is either a fragment or an activity
- Similar to UIViewController in Xcode
- Both can have a layout, managed in an xml file
- An app needs at least one activity. Fragments are optional

ACTIVITY VS. FRAGMENT



IMPORTANT ACTIVITY METHODS

- onCreate(Bundle) called to initialize the activity. Call setContentView with a layout resource defining your UI.
- onPause() called when the user is leaving the activity. Execution is very brief, therefore, don't save application or user data.
- onStop() called when the activity is left. Use this method (and not onPause) to persist user data (check here).

Hint: There are scenarios where only onPause is called but not onStop, ie an incoming notification.

ACTIVITY LIFECYCLE

FRAGMENT

Fragments are:

- building blocks (small UI components)
- used to encapsulate UI components of an activiy
- managed by the owning activity
- easily reused
- typically reordered to support different devices (phone / tablet).
  http://developer.android.com/guide/components/fragments.html

FRAGMENT LIFECYCLE I

Android recommends to implement at least the following three methods in each fragment:

- onCreate: Called when the Fragment is created (only once within a lifecycle). Create all components that should be retained during Pause/Resume.
- onCreateView: Called when the Fragment is drawn. Return the Fragment layout.
- onPause: Called when the Fragment is left by the user. In contrast to an Activity, use this method to persist your data (see here).

Use onViewCreated to initialize the components.

FRAGMENT LIFECYCLE II

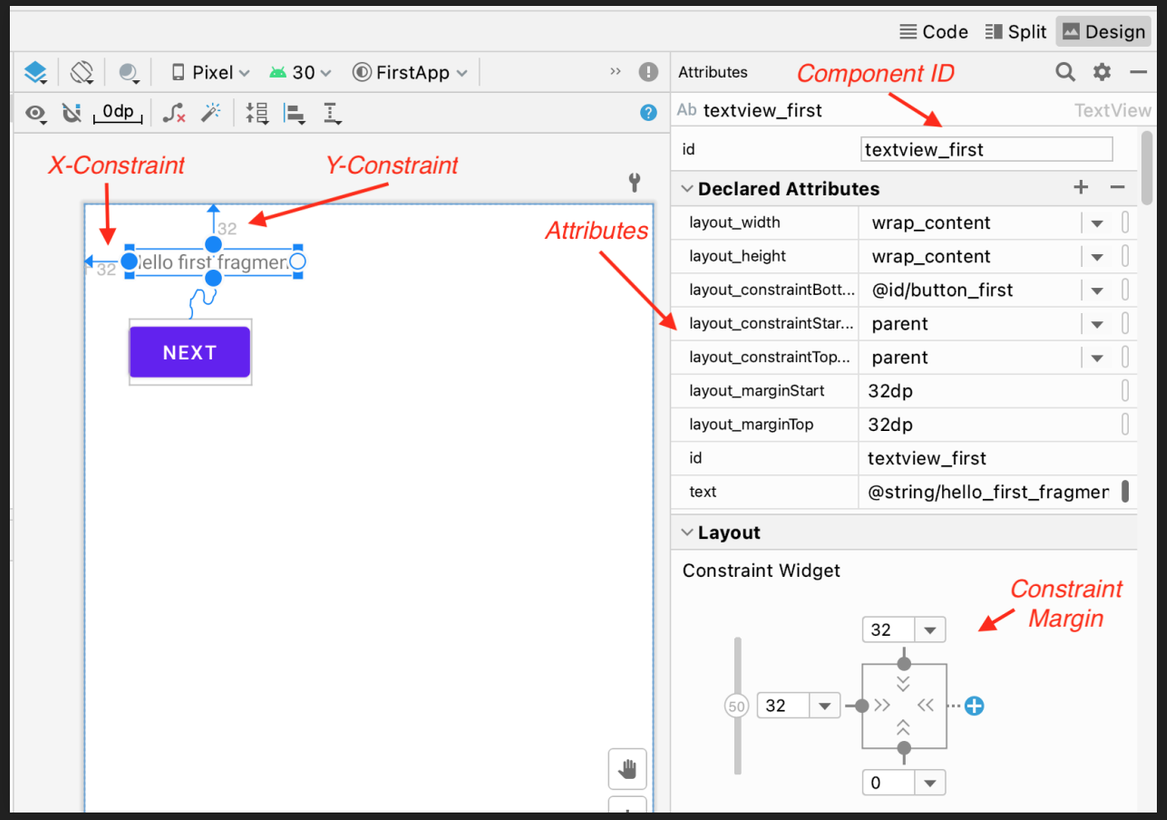
Source: developer.android.com

BASIC COMPONENTS

- All components can either be created in code or in the layout xml.
- Components, that are created in layout xml will be available in code using view binding.
- All components inherit from android.view.View
- Some components can be nested.

COMPONENT PLACEMENT

- ConstraintLayout is Android's standard layout
- All components must have at least two constraints.



VIEW BINDING I

Use View Binding to access components in code that have been defined in layout:

```
//binding stores the reference to the layout, the class
//FragmentFirstBinding is generated automatically
private var _binding: FragmentFirstBinding? = null

//in order to address the binding without an optional,
//Android creates another property binding (without the
//underscore) and assigns it the extracting reference
private val binding get() = _binding!!
//of course, this will result in a NullPointerException
//if it is called before onCreateView or after onDestroyView
```

VIEW BINDING II

```
//in onCreateView, the binding is initialized:
override fun onCreateView(...) {
//store the result of inflate in binding
 binding = FragmentFirstBinding.inflate(inflater, container, false)
 return binding.root
}

override fun onViewCreated(...) {
//now use binding to access the layout components
//The name toolbar matches the id
binding.textView.text = "A text"
}
```

VIEW BASE CLASS

- The view base class provides methods for rendering and event management.
- View is the base class for certain components like TextView or EditText, the ViewGroup subclass is the base class for layout components (nested components).

IMPORTANT VIEW METHODS AND ATTRIBUTES

- clickable: Does this component react to click events?
- elevation: Z-Index of the component
- onClick: Listener for the onClick event
- padding, margin: Padding and margin for better layouting
- width, height: Width and Height of the component

TEXTVIEW I

TextView can be used to show text to the user. It is possible to:

- change the size, style and color of the font to use.
- make the TextView selectable (for copying).
- use multiple lines when displaying the text.

TEXTVIEW II

```
<TextView
    android:id="@+id/result"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="20dp"
    android:text="Hello World!" />
```

Get the component in code:

```
result.text = "New Text"
```

TEXTVIEW III

```
//by default, the TextView is showing the complete text
//regardless of the length. You can let Android cut the
//text (result is a TextView component)

//maximum numbers of lines to display
result.maxLines = 1

//truncating the text at the end (showing three dots)
result.ellipsize = TextUtils.TruncateAt.END
```

EDITTEXT I

EditText is a text input component:

```
    <EditText
        android:id="@+id/first_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="First Name"/>
```

Get the component in code:

```
val name = firstName.text.toString();
```

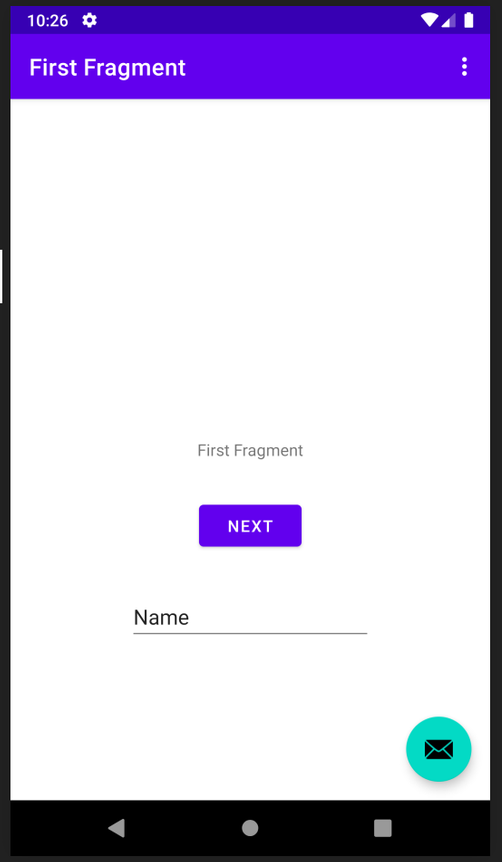
EDITTEXT II

EditText provides a number of interesting attributes:

- android:inputType: Customize the input type (ie. textPerson, textPassword or phone)
- android:maxLines: Set the number of lines
- android:digits: Set the allowed digits (ie "012") if the inputType is number

EXERCISE

Your project should already have a "First fragment" TextView and a button "NEXT". Add an EditText underneath the button.



BUTTON

A Button can be pressed to perform an action:

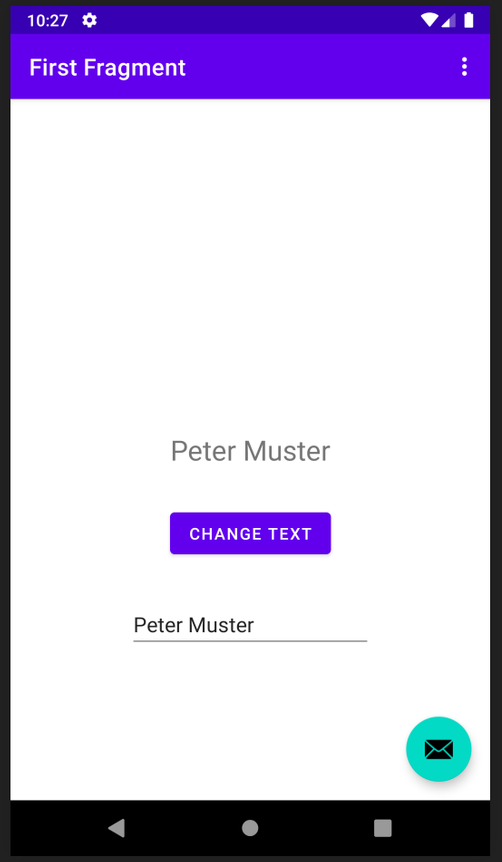
```
<Button
    android:id="@+id/ok_button"
    android:layout_marginTop="20dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="OK"  />
```

Set the action in code:

```
ok_button.setOnClickListener { view ->
      Log.i("TAG", "button clicked");
}
```

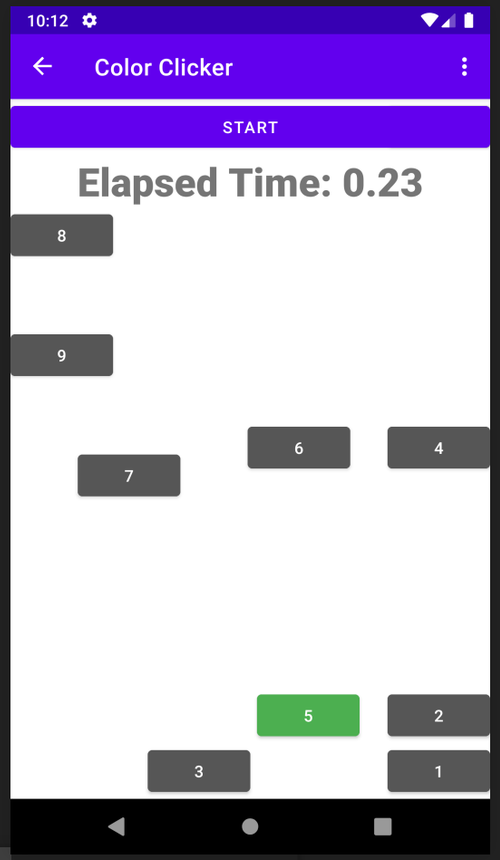
EXERCISE

Adapt the OnClickListener of the button:
Read the text from the EditText and copy it to the TextView.



EXERCISE: COLOR CLICKER

Write a ColorClicker App: Place 9 buttons randomly on the surface (but still check that constraints are set!). After the user has clicked "START", one button is randomly chosen and the background color put to green. The user must click the chosen button fast. Show the elapsed time in a TextView:




```
//random number between 1 and 9 (inclusive)
colorButtonNum = Random.nextInt(1, 9)
//get time in millis
val time = System.currentTimeMillis()
//converting double "d" to String with two decimal places
val dStr = "%.2f".format(d)
//set the background color of a button
button.setBackgroundColor(Color.parseColor("#F6F6F6")
```

HIDE KEYBOARD

```
//by default, the user can show or hide the keyboard
//using the bottom left arrow button. If the keyboard
//should be automatically hidden, use the following method
fun hideKeyboard() {
  val manager: InputMethodManager =
   requireActivity().getSystemService(Context.INPUT_METHOD_SERVICE)
       as InputMethodManager
  if (manager != null) {
    manager.hideSoftInputFromWindow(
     requireActivity()
     .findViewById<View>(android.R.id.content).windowToken, 0)
  }
  binding.editText.clearFocus() //remove focus from EditText
}
```

EXERCISE

Develop an Android calculator. You can choose between two versions:

Simple (left): two input fields, buttons for selecting the operation and a TextView that shows the result.

Advanced (right): simulate a solid-state electronic one.