# Android Intro

with

*Simon Arneson*,
Senior Software Developer

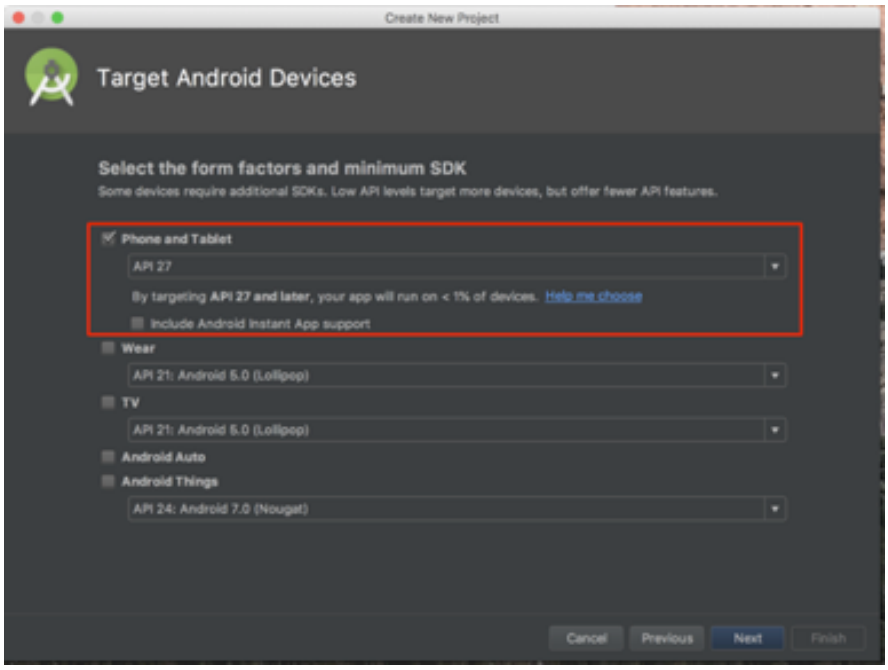
devies

We are developers. We are devies

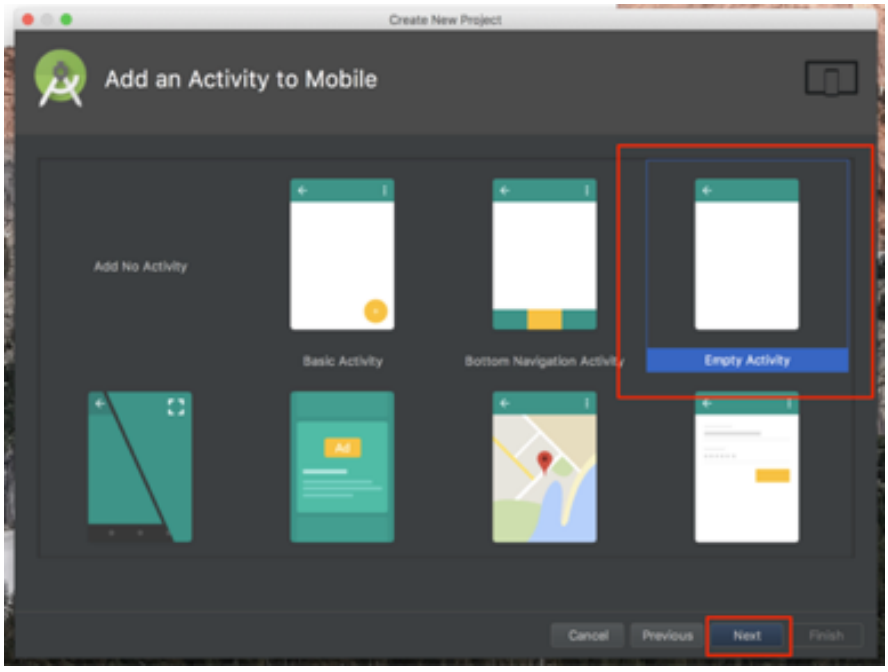
# Android Intro

## Create project

1. Open Android Studio
2. Start a new Android Studio project, give it the name "ExampleApp"
3. Uncheck include Kotlin support
4. Select API level 27 under "Phone and Tablet




5. Select "Empty Activity"

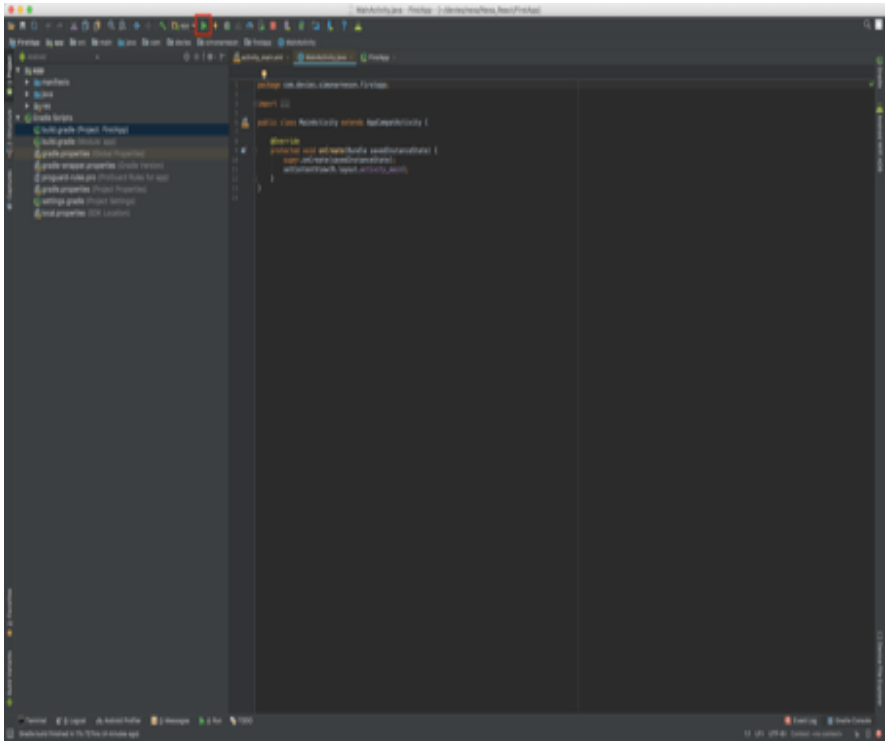



6. Click "Finished"

# Android Intro

## First run

1. Run app



## Create an emulator

1. Add Emulator
2. Download SDK
3. While waiting, browse app folder to find main xml layout file, build.gradle for your project android manifest
4. Select SDK version
5. Name emulator
6. Click ok and let your app start!

# Android Intro

## Let's make some changes

Take a look at MainActivity.java bit-by-bit:

1. First, you override onCreate(), one of the methods available to you in an Activity. It runs when your Activity is created, giving you ample time to set up anything you need. This is often the first method you create for an Activity. (refer to Android life cycle diagram, last page of this document)

2. You then call the superclass implementation of onCreate() to ensure any set up required for the Activity is performed. This is a required step. If you don't do it, your app will crash with an exception warning.

3. Finally, you tell your Activity to show a screen by using setContentView(), passing in a layout that's referenced from R.

Note: R is a special class generated by Android for your app's various assets. These could be a screen layout like you see above, or it could be something like a localized string, image or animation. Android generates R while building, so you shouldn't modify it at all. More information about it is available from developer.android.com.

## Modify a view

To start of, let's something in the view from the code-side of things. We will now change the text shown in the text view in the app.

1. Set an id for the TextView, ex. myAmazingTextView which currently says "Hello World!", either using the GUI editor or by editing the xml.
android:id="@+id/myAmazingTextView"

2. Open MainActivity.java and use the method findViewById() to fetch a reference to the TextView, save the reference in a private instance variable of type TextView.
public <T extends View> T findViewById(@IdRes int id)

3. Set the text of the textview to "I changed programmatically" using the method setText found on the TextView instance.
public final void setText(CharSequence text)

4. Run the app, the text displayed show now be different than before.

# Android Intro

## User input

An app would in general not be very interesting if the user did not have any power over it.
In this step we will let the user enter a text string. If the user then taps the button the text will be shown on the screen.

## GUI

1. Open main_activity.xml and edit it in 'text mode'.

2. Change the parent layout from ConstraintLayout to RelativeLayout

3. Remove the constraint tags from the TextView

4. Center the text view in it's parent using the centerInParent tag
 android:layout_centerInParent="true"

5. Add an EditText element below the text view and position it.
try layout_below

6. Set the layout_width to match_parent and layout_height to 40dp.

7. Give the element an id.

8. Add a Button element below the EditText element.

9. Set the layout_width to 150dp and layout_height to 60dp.

10. Give it an id.

11. Center it horizontally and place it below the amazing edit text

12. Change the button text to "Do things!"

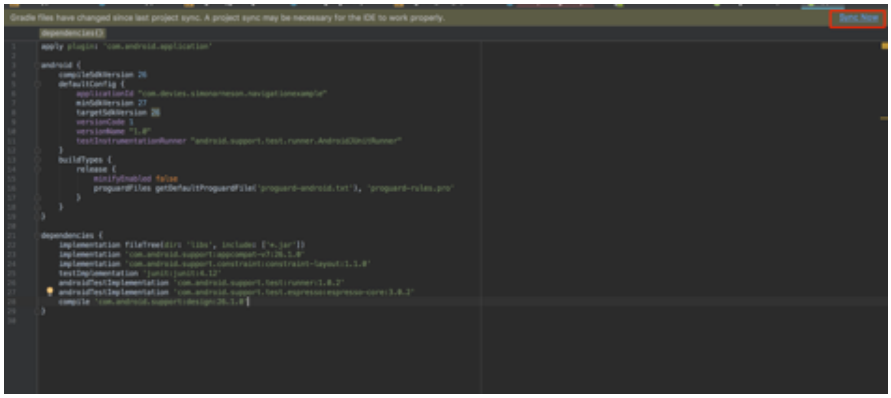
# Android Intro

## Code

We will now set up the application logic to read the input data and set it as text on the text view when the user taps the button.

### Setting up a click listener

1. Open MainApplication.java
2. Create private instance variables to store references to the Button and the EditText elements.
3. Use findViewById to set these references in the onCreate method.
4. Create a local View.OnClickListener called listener and assign a new View.OnClickListener to the variable, make sure it generates an override of the onClick method.
5. For the next step we need to add a dependency to our app. This is done by opening the build.gradle file connected to your app, (marked Module: App) and add a row under dependencies:
compile 'com.android.support:design:26.1.0'
6. Gradle is now outdated. Click Sync Now in the top bar message showing.



7. In the implementation block of the onClick method create a Snackbar message using Snackbar.make(view.getContext(), “Hello”, Snackbar.LENGTH_LONG)
ASnackbars provide lightweight feedback about an operation. They show a brief message at the bottom of the screen on mobile and lower left on larger devices. Snackbars appear above all other elements on screen and only one can be displayed at a time.

They automatically disappear after a timeout or after user interaction elsewhere on the screen, particularly after interactions that summon a new surface or activity. Snackbars can be swiped off screen.

8. Set the onclick listener on the button by using the setOnClickListener method on the button instance.
9. Start the app and click the button. Why does nothing happen?
Try using Android Studio Intellisense to help you out.
10. Add the missing code and run the app again. The button should now show a Snackbar message when clicked.

# Android Intro

## Reading and modifying values

1. Remove the Snackbar-code from the onClick method.
2. Create a string type variable called userInputText and set its value to “Hello ”.
3. Fetch the text from your EditText component using the method getText on the EditText instance and get the String representation of the Editable by using ToString.
4. Concatenate the value from the EditText to the current value in userInputText using the + operator and store it in userInputText. (stringA + stringB).
5. Use the setText method on the TextView instance to set the text of the TextView to the text the user has entered.
6. Run the app, enter something in the EditText element, click the button and ensure the TextView text is updated.

# Application flow

An application with just one view is obviously no fun. As discussed earlier navigation and application flow can be solved using a number of patterns. Examples are using multiple activities, single activity with multiple fragments and last but not least the fairly new architecture components.

Currently the architecture / navigation pattern most commonly used is the second alternative: Having one activity and switching between multiple fragments and or views. Architecture components still have not properly ‘landed’ but will probably be more and more common the coming years. See https://android-developers.googleblog.com/2018/05/use-android-jetpack-to-accelerate-your.html for more information.

What we will do in this step is instead of showing the greeting immediately in the same view we will navigate to another view and show it there.

# Android Intro

## Single activity application

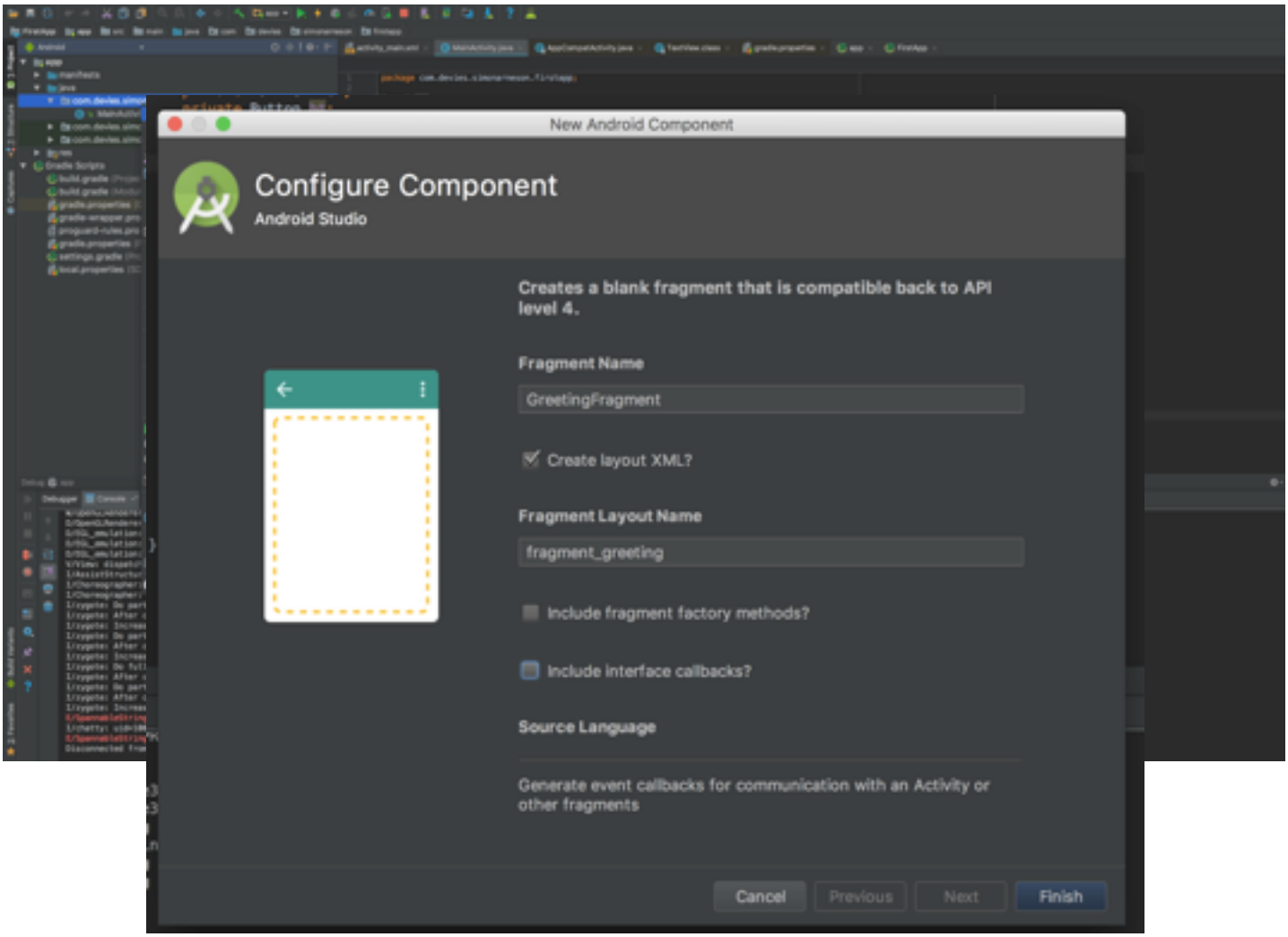
## Start all over

1. Create a new project following the steps from the start of the tutorial. Give the project the name NavigationExample.

## Add a new Fragment

We'll start of by creating a new Fragment.

1. This is most easily done by navigating to your package in the left side menu and right clicking the folder. (app/java/your.package.name)
2. A context menu will show from which you select New >Fragment > Fragment(Blank)
3. Give your new fragment the name GreetingFragment




4. Deselect to options 'Include fragment factory methods' and 'Include factory callbacks' and click finish.
5. Repeat step 2 - 4 and create another Fragment called UserInputFragment.

# Android Intro

## Separate / move the code

We now have two fragments and a corresponding xml layout file has been created for each one.
(Check the res/layout folder).

Our first step now is to save some time by reusing our layout from the first app to the UserInputFragment.

1. Start by copying and pasting the xml from activity_main.xml (from the first example app) into fragment_user_input.xml
2. Remove the tools:context tag from the root element.
3. Make sure to remove the TextView element and modify the xml to center the EditText vertically again.

## Include a Fragment in an Activity

To include a fragment inside an Activity from the start, one option is to add it to the layout xml of the Activity.

1. Open activity_main.xml.
2. Remove the TextView and change the type of the root element to RelativeLayout
3. Inside the RelativeLayout (where the TextView used to be) add a Fragment element.
4. Set the required properties width and height to match the parent.
5. Set the android:name property to match the full name and package name of your UserInputFragment.
6. Set the android:id property to "@id+/main_fragment"
7. Start the app. It should now visually look a lot like your first app.

## Differences between Fragment and Activity

1. Open UserInputFragment, fetch a reference to myAmazingButton and add an OnClickListener to the button the same way as in the first example app.
However, this being a fragment forces us to do some things a little bit differently.
   a. Instead of coding in the onCreate method, we code in onViewCreated
   b. The method findViewById is not available on Fragments as it is on Activities, we need to fetch a reference to the main view of the Fragment and call the method from there getView().findViewById
2. To make sure everything works. Add a snackbar message in the onClickListener and run your app.

# Android Intro

## Switching between fragments

To manage the fragments in your activity, you need to use FragmentManager. To get it, call getSupportFragmentManager() from on your fragment.

We have problem however. Fragments added in the xml layout cannot be replaced. We need to refactor our Activity to create the fragment from the code side of the app.

### Refactoring

1. Remove the android:name tag from the fragment element in activity_main.xml and change the fragment element to a RelativeLayout.
2. Create a new UserInputFragment in the onCreate method of MainActivity.
3. Get the FragmentManager and get a FragmentTransaction by calling beginTransaction on the FragmentManager.
4. Replace the view in the layout which has the tag main_fragment with the new fragment by using the replace method on the FragmentTransaction instance.
5. Finish up by calling commit on the FragmentTransaction to make your changes permanent..
getSupportFragmentManager().beginTransaction().replace(R.id.main_fragment, startFragment).commit()
6. Start the app, it should now be back where we started before the refactoring.

### Continue doing fun code

We are now ready to code the fragment switching.

1. Create an instance of the Fragment we want to navigate to, GreetingsFragment, in the onClick method in the UserInputFragment.
2. Get the FragmentManager by this time using getFragmentManager (as we are in a fragment already) and get a FragmentTransaction by calling beginTransaction on the FragmentManager.
3. Use the replace method on the fragment transaction just like we did on onCreate in MainActivity, but this time with the fragment we want to navigate to.
4. Start the app. Click the button, the app should now navigate to the GreetingFragment.

# Android Intro

## Passing data between fragments

The greeting right now is not very personal, and the functionality is the same as in out first example app.
We need to set the text on the TextField in the GreetingsFragment but the data is collected in the UserInputFragment.
This forces us to somehow pass data between the fragments. Data management / global state is pretty complicated in the Android world, but we will use a simplified easy way to pass data for now.
This can be done by sending arguments in the Bundle.

### Send data

1. Start by creating a new Bundle and storing it in a local variable called arguments. Do this immediately after you create the GreetingsFragment.
2. Use the putString method on the Bundle instance to add the value from the EditText element (Peek at the first example app to remember how we fetched the data, remember we are now in a Fragment). As arguments is a kind of key-value store we need to provide a key for our data. We will use the key "name" for this example.
3. Finish off by setting the arguments on the fragment we are to switch to using the method setArguments on the GreetingsFragment instance and pass in our Bundle instance.

### Read data

In the fragment we switch to, we now need to access the name we sent from the previos fragment. This can be done in the lifecycle method onCreate by:

1. Override the method onCreate(@Nullable Bundle savedInstanceState) in GreetingFragment.
2. Getting the arguments using the method getArguments available in the fragment. Store the data in a local Bundle variable called arguments.
3. Get the data using the method getString(String key) on the arguments using the same key ("name") we used when sending the data.
4. Set the string to the TextField the same way we did in the first example.
5. Start the app, enter your name, click the button and enjoy your First Amazing App.

### Just kidding

As you probably noticed, something is fishy. It seems as if the view we are trying to find does not yet exist.
Refer to the Android Fragment Lifecycle and try to figure out what is wrong.

# Android Intro

## Animate the transition

The transition between the fragments does not look very god at the moment, to do something about that we will add some custom animation. This can be come by calling setCustomAnimations on the transaction before calling replace.
This method takes two parameters, the animation used by an entering fragment, and the one used by an exiting fragment.
We provide these in form of animations defined in xml files in the res folder.

1. Create a new folder called anim under the res folder of your app.
2. Add the following to files / snippets to the anim folder:

res/anim/slide_in_right.xml
res/anim/slide_out_left.xml

Below is an example of slide_in_right.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set>
  <translate xmlns:android="http://schemas.android.com/apk/res/android"
    android:fromXDelta="100%"
    android:toXDelta="0"
    android:interpolator="@android:anim/decelerate_interpolator"
    android:duration="500"/>
</set>
```

Try to figure out what slide_out_right.xml is supposed to look like.

You are now done!
Congratulations!

Simon Arneson
arneson@devies.se