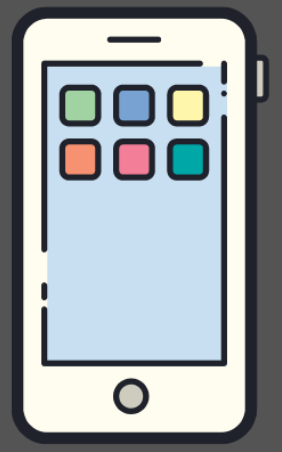


MOBILE DEVELOPMENT

Activity Overview

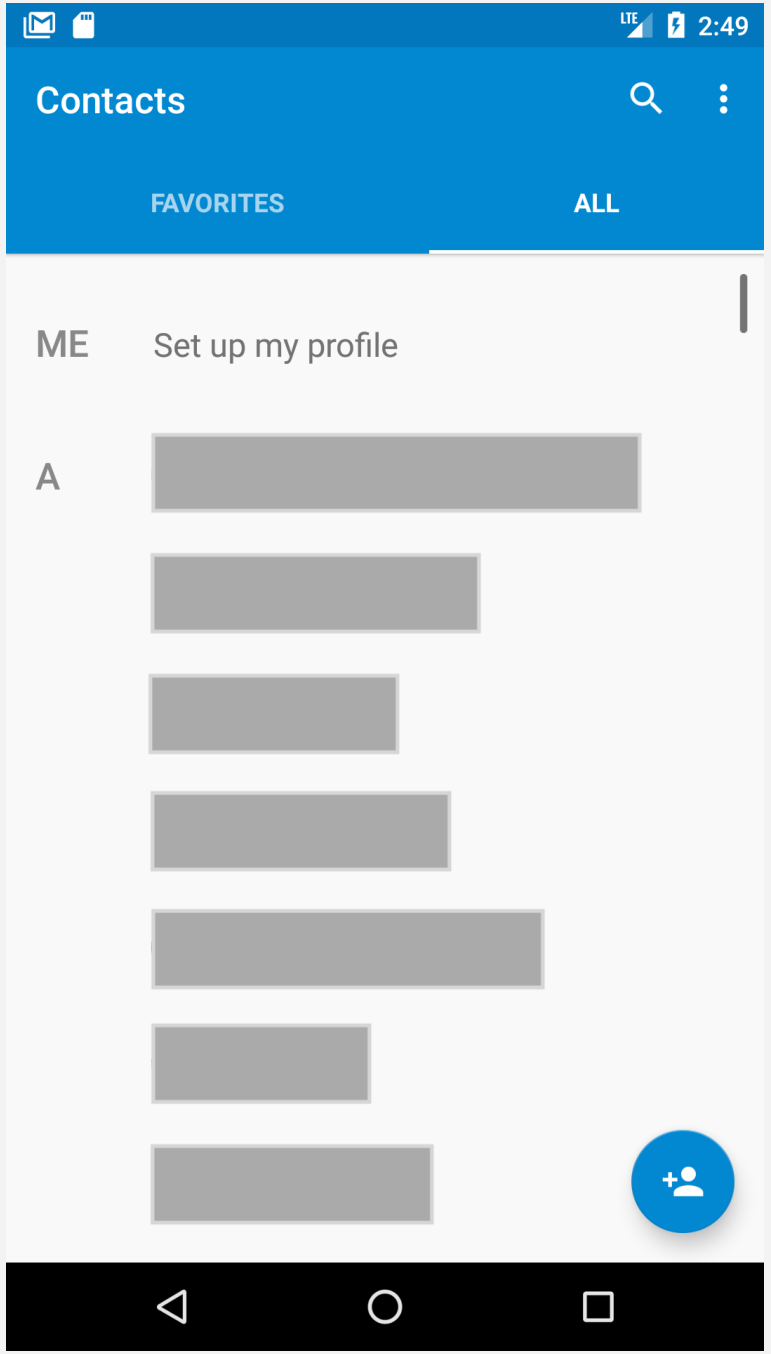
Outline

- Motivation
- Defining an Activity
- Defining Context
- Handling Multiple Activities
- Passing Data Between Activities

Open your Contacts App

What **functionalities** can you identify?

Don't worry about formalities, just try to identify what functions you think it has ☺



(Some) Contacts Functionality

- List of Contacts
- Add Contact
- View Contact
- Edit contact
- Delete Contact
- Call Contact
- Search
- Settings
- Export / Share Contact
- Manage Account
- Block Numbers

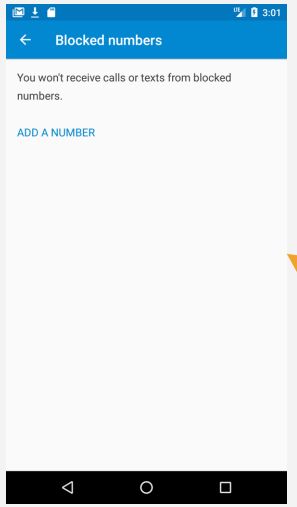
Reflect: How did you identify functions?

Streamlining Contacts Functionality…

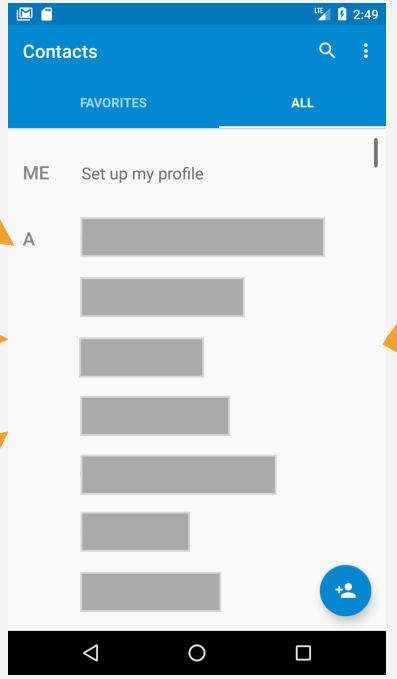
- View All Contacts (Home)
  - View favorites
  - Search
  - Share contact
- Add+Edit Contact
- View Contact Details
  - Delete contact
  - Share/export contact
- Manage Settings
- Manage Account
- View Block Numbers

Most functionalities are contained in their own screen. If a screen has more than one functionality, its because the functions complement each other, like how searching makes viewing easier.

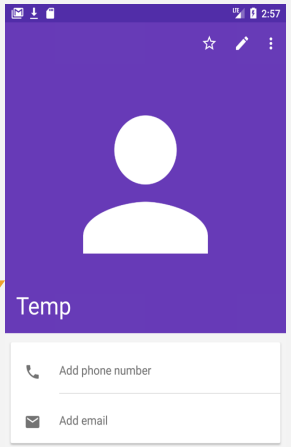
Block Numbers



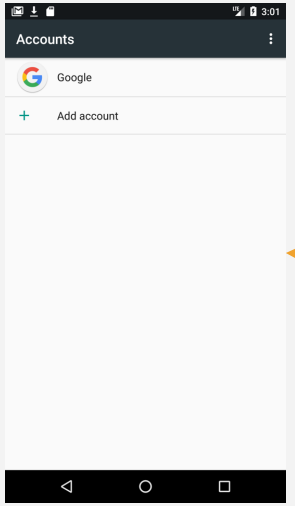
Main Screen



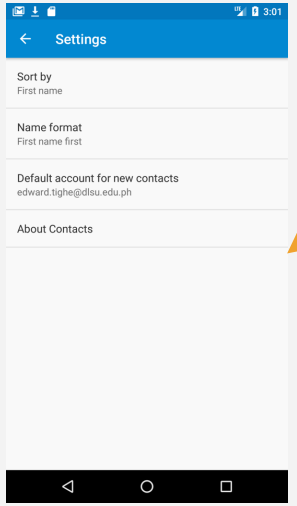
View Contact Screen



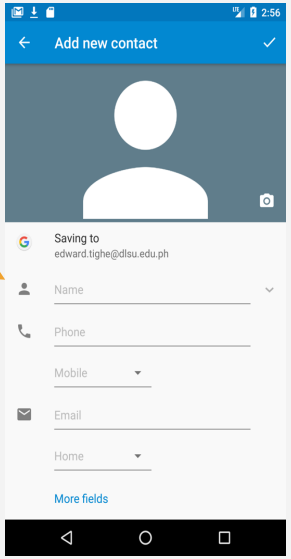
Manage Accounts



Settings



Add/Edit Contact Screen

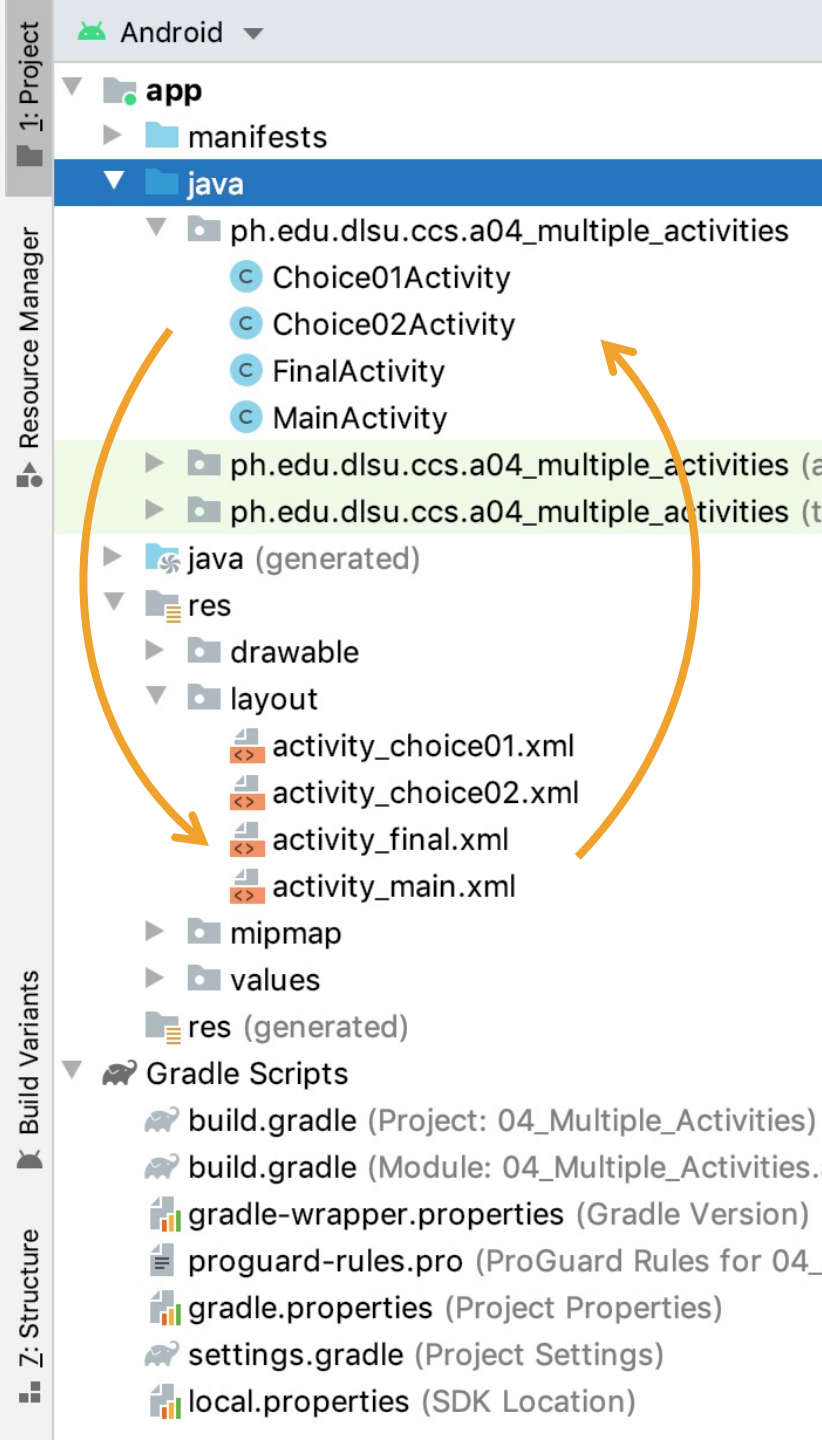


Some screens are reusable (e.g. Add/Edit), while other screens are of different apps (e.g. Manage Accounts).

What is an **Activity**?

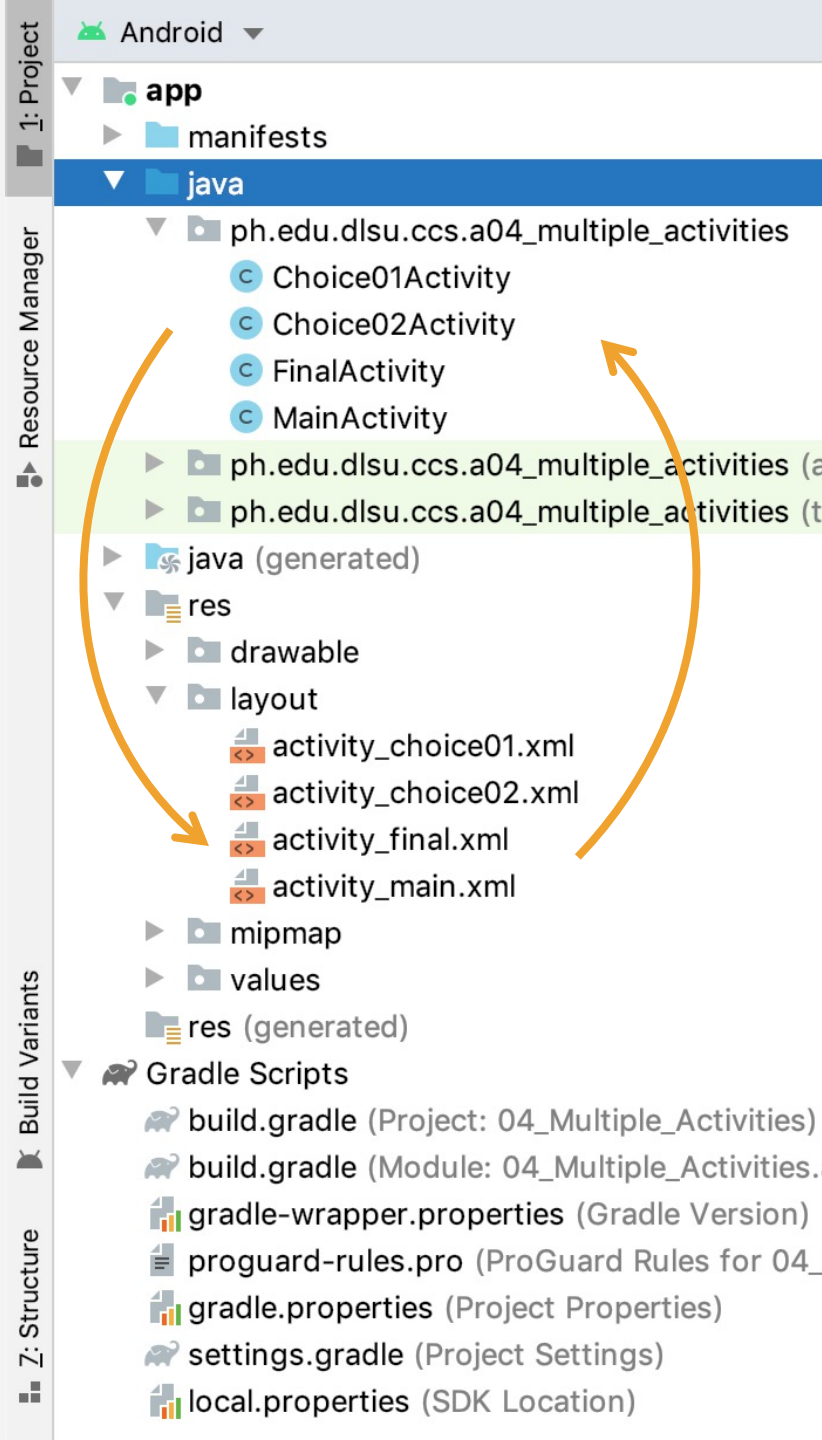
What is an **Activity**?

- **Basic component** of an Android Application
- Provides a **framework** for the code to interact with the layouts
- Usually associated to a layout file, which it handles inflating



What is an Activity?

- Usually thought of as a single screen
  - Single screen == Single task
    - Or at least a set of related tasks…
  - Hence, a major task of an app can be the basis for an activity
- Functions very much like a Controller
  - C in MVC
  - In charge of handling events and displaying/passing data



```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ph.edu.dlsu.ccs.a04_multiple_activities">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".Choice01Activity" />
        <activity android:name=".Choice02Activity" />
        <activity android:name=".FinalActivity"></activity>
    </application>

</manifest>
```

Notice…

- There is no main function for Android apps
- The Android system lunches a main activity as defined in the Manifest

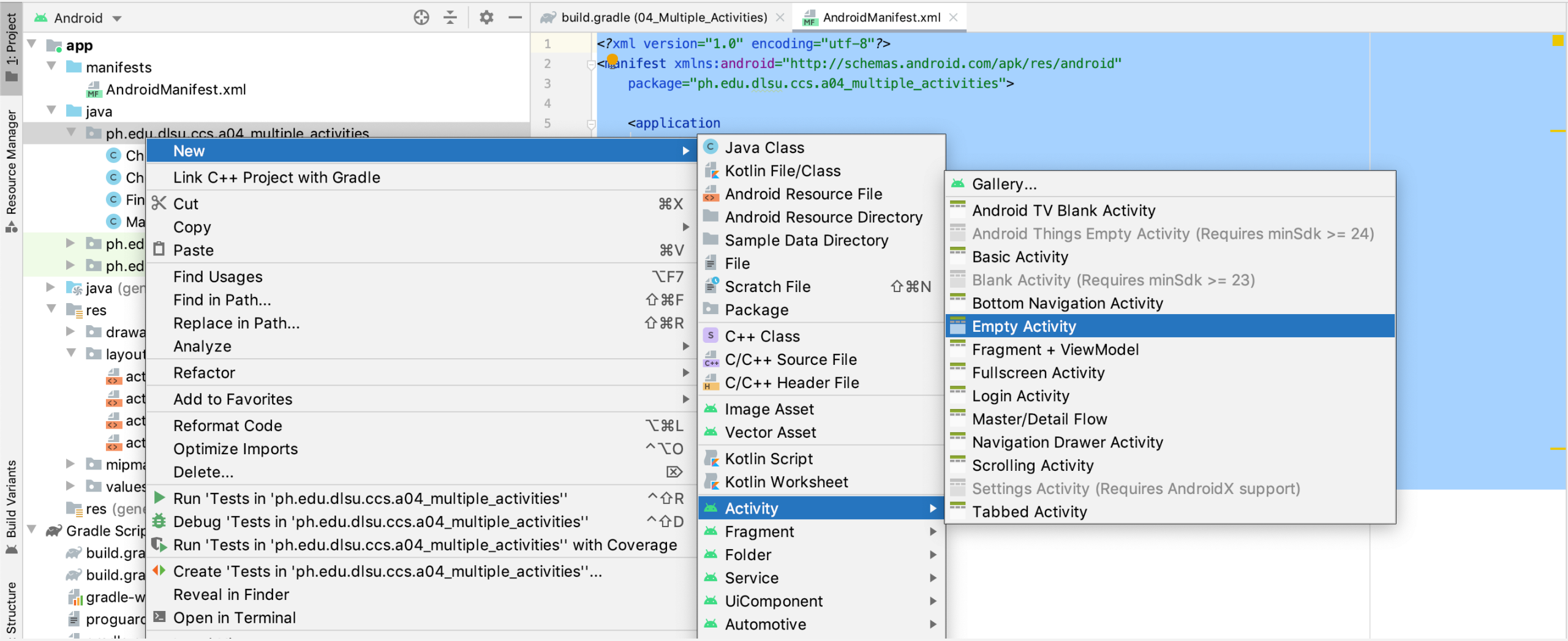
Main -> Define an activity as an entry point to the app
Launcher -> Indicates an activity as part of the system's app launcher

Creating an Activity

Activity

- When creating an activity by default…

By default, I mean by allowing Android Studio to do so...



This way, Android Studio automatically adds a layout file and an Activity details needed in the Manifest. You can create an Activity from scratch, but make sure to create a layout file and define the activity in the Manifest or else Android won't recognize it.

Activity

- When creating an activity by default, notice that an activity is a class that extends AppCompatActivity
  - AppCompatActivity is a subclass of Activity
  - Compat -> Compatible – implying this will work on older devices

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
}
```

Activity

- Additionally, by default, Activities are associated to a Layout (and it resources) through setContentView()
  - Can only have one ContentView at a time
  - Typically, you'd have one activity associated to one layout
  - Multiple activities could use the same layout [if reusable]

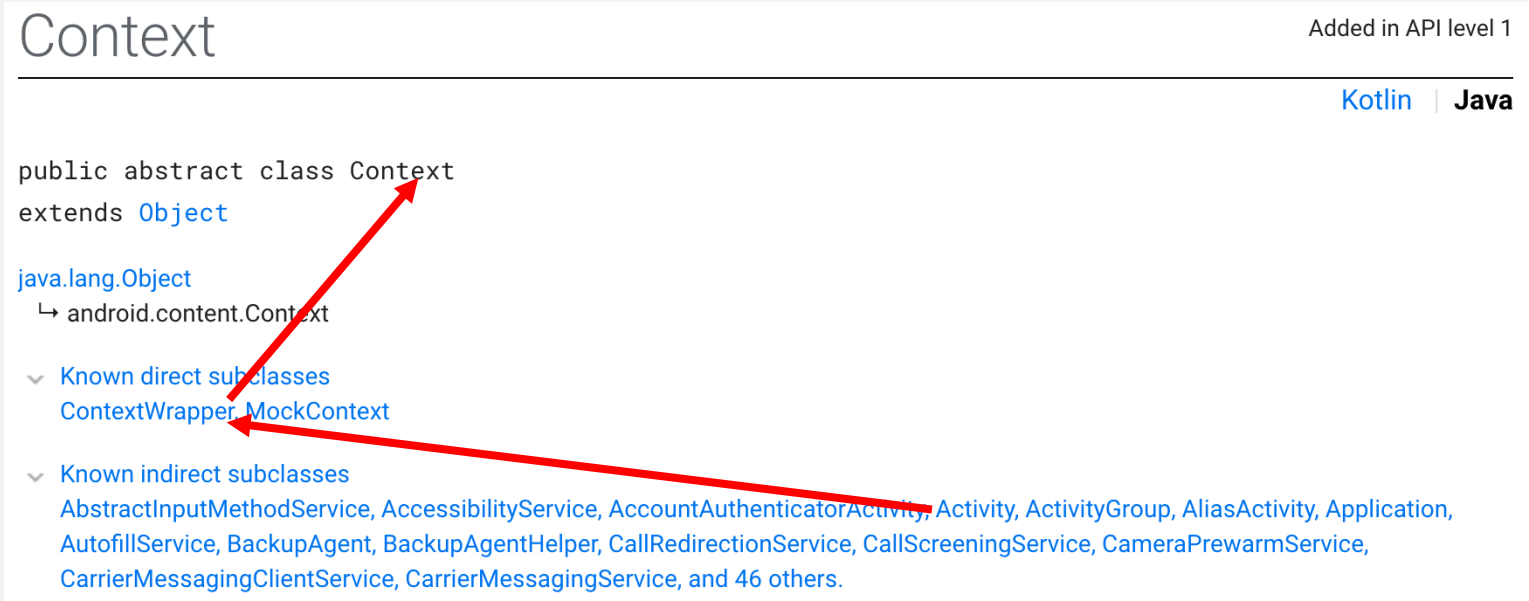
Just like what we did with a ViewHolder's View in the RecyclerView, setOnContactView() **inflates** the provided layout

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
}
```

If the ContentView isn't properly set up, your Activity won't be able to access the the layout properly

Activity → Context

- If you traces the superclass of Activity, you'll find that an Activity is also Context
- Context is an interface to global information about an application | activity (i.e. it's environment)
  - It allows access to resources and classes



```java
public class MainActivity extends AppCompatActivity {
    private EditText first_name_et;
    private EditText last_name_et;
    private Button add_btn;
    private LinearLayout name_list_vll;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        this.first_name_et = findViewById(R.id.firstname_etv);
        this.last_name_et = findViewById(R.id.lastname_etv);
        this.add_btn = findViewById(R.id.add_btn);
        this.name_list_vll = findViewById(R.id.namelist_vll);

        this.add_btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                TextView temp_tv = new TextView(MainActivity.this);
                temp_tv.setText(last_name_et.getText() + ", " + first_name_et.getText());
                name_list_vll.addView(temp_tv);
            }
        });
    }
}
```

RECALL

When creating Views, we need to pass it a Context object.

There are a few ways to reference a Context object:

- Activity.this (the activity's context)
- View.getContext() [the View's associated context, whatever that may be]
- getApplicationContext() [the context of the entire application, not just the activity]

In this case, its actually better to use MainActivity.this or view.getContext() – not getApplicationContext().

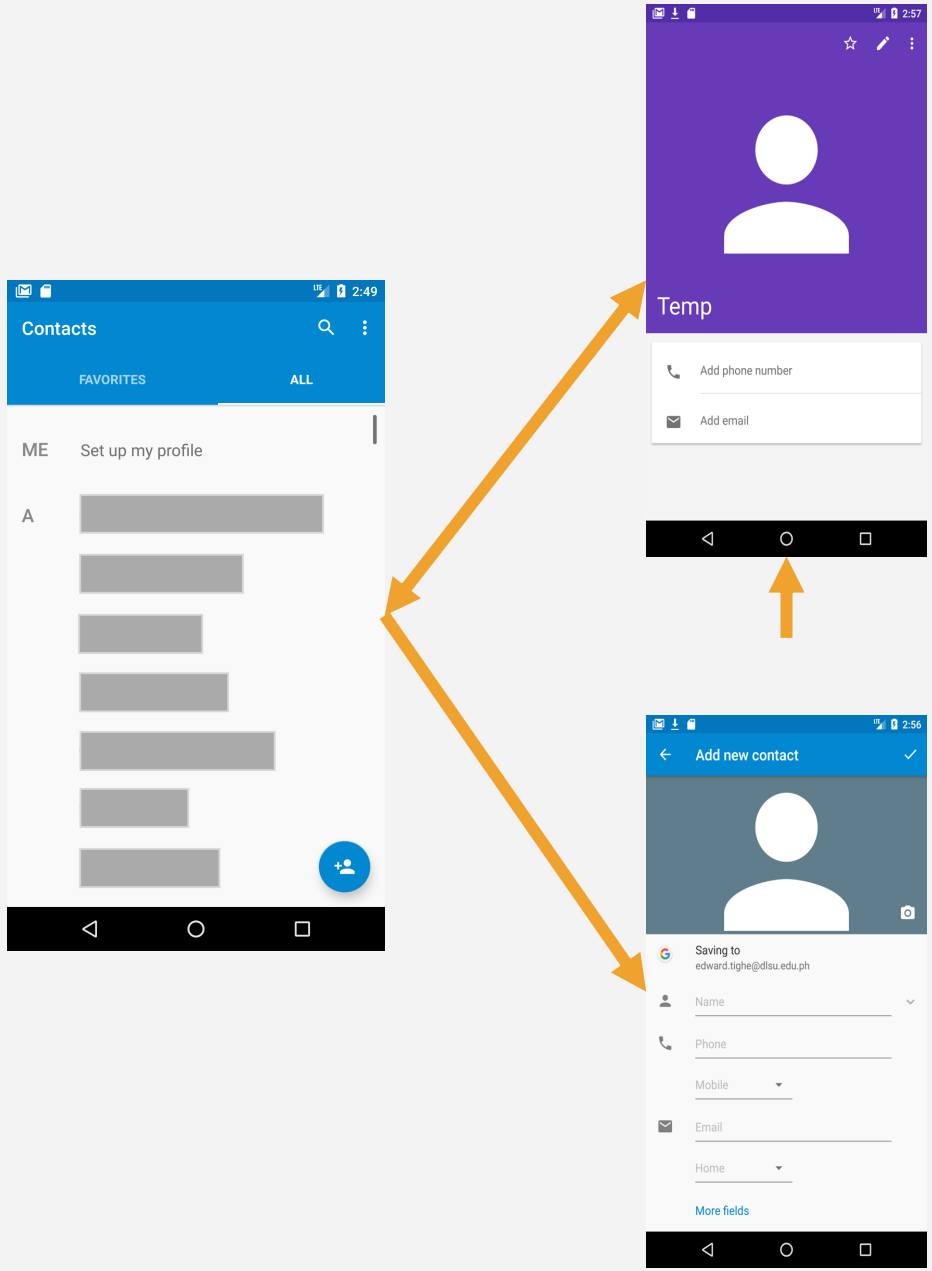
Rule of thumb is that if the object that required context is centered on the activity (like Views), use the Activity's context. Otherwise, use the Application's context. ☺

Questions so far?

Multiple Activities

- Simple applications might only need one activity, but this is rarely the case
- Most apps have multiple activities with each corresponding to a specific functionality

This implies that we'd need to to link our Activities together somehow…



Moving from Activity to Activity

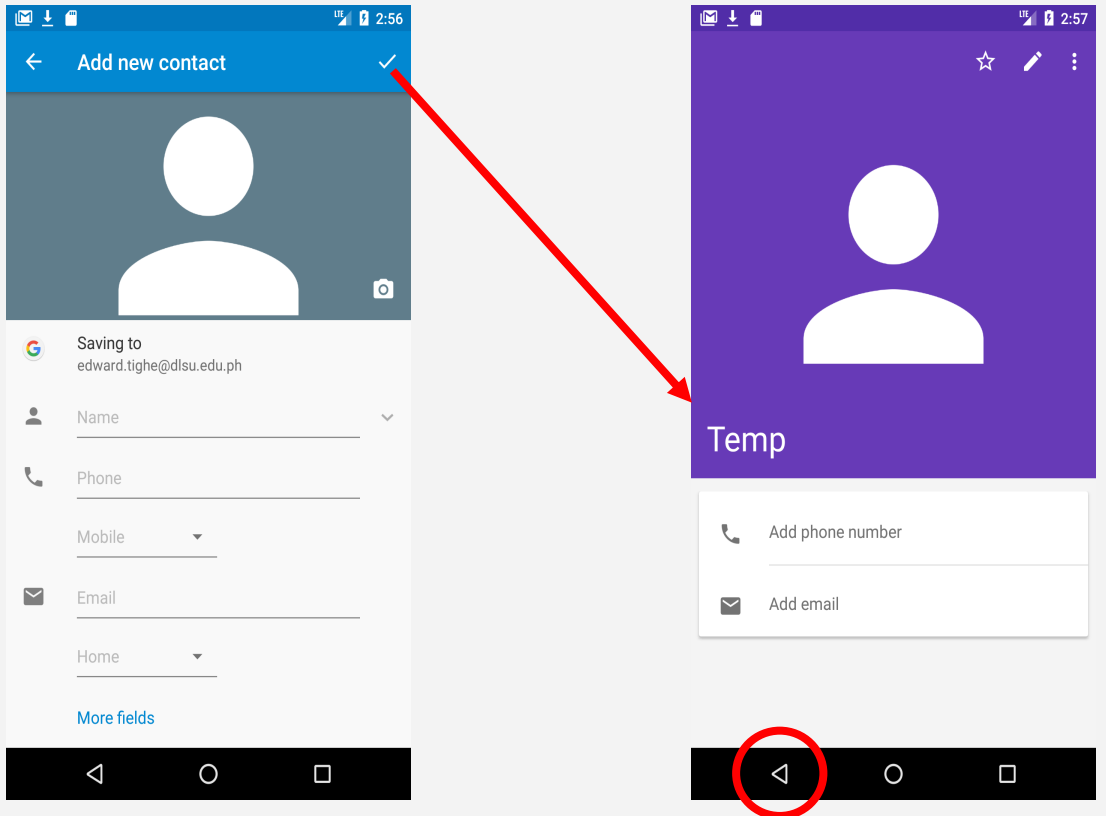
- In order to move to another Activity, we don't instantiate the second activity
- Instead, we send a "message" to the system and request for the other activity to be launched
  - For this, we'll need to create + instantiate an Intent object and request for that an activity be started, as such:

```
Intent intent = new Intent(SourceActivity.this, DestinationActivity.class);
startActivity(intent);
```

Context

Moving from Activity to Activity

- Once in the new activity, you can return to the previous activity by clicking the back button

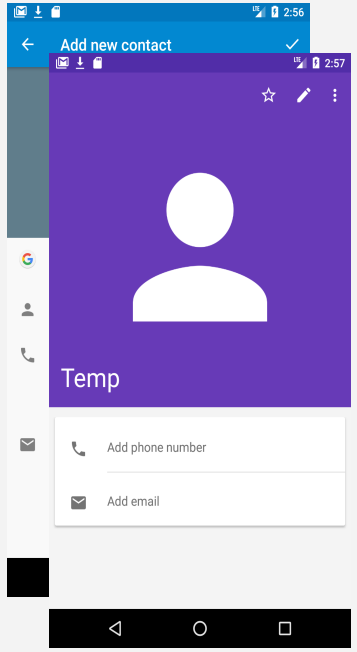


Design Tip:

Avoid creating a back button if all it does is redirect to the previous screen. If its logic mimics the system's back button, there's no need to include it.

Moving from Activity to Activity

- When using startActivity(), your app creates the destination Activity and moves that to the front of the Activity stack

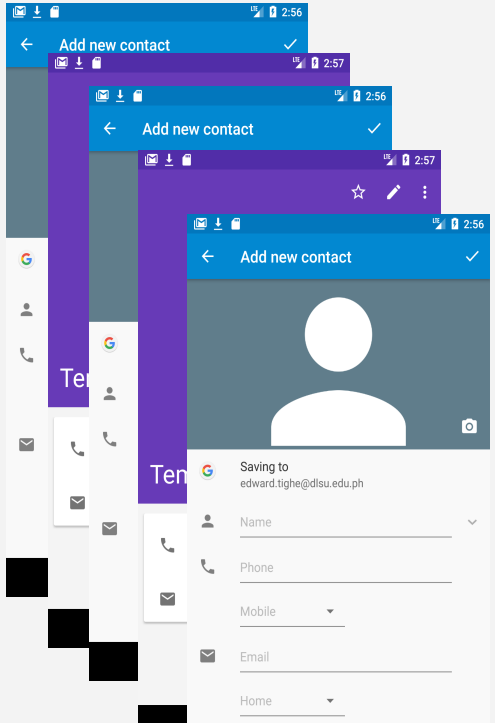


In the visualization to the side, we see that the active Activity (on top of the stack) is the View Contact screen. This is where the user currently is.

If we click the system's back button from this screen, the Android system would "destroy" the current activity and return focus to the previous activity still in the stack.

Moving from Activity to Activity

- Don’t abuse startActivity()!
  - It is possible to cycle between Activities that don’t really need to return to each other’s previous states



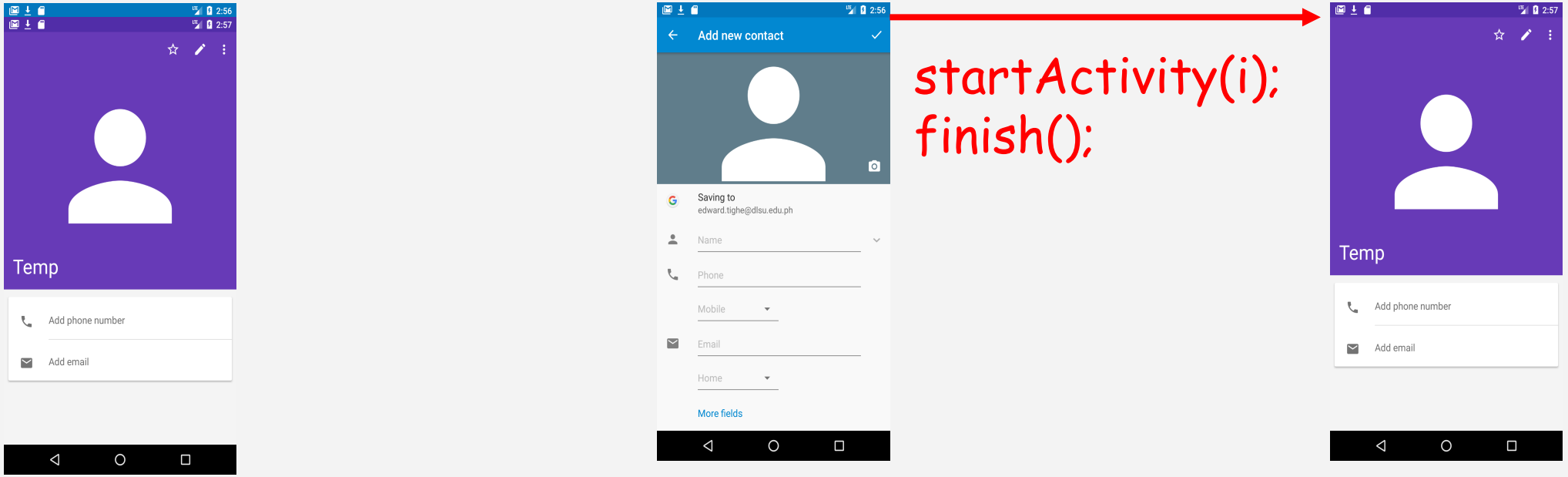
For demonstration's sake, the Activity stack presented to the side is a result of having two activities constantly use startActivity

Imagine being the user and pressing the back button – you'll basically be cycling through all the previously started activities… and that would be pretty annoying.

Moving from Activity to Activity

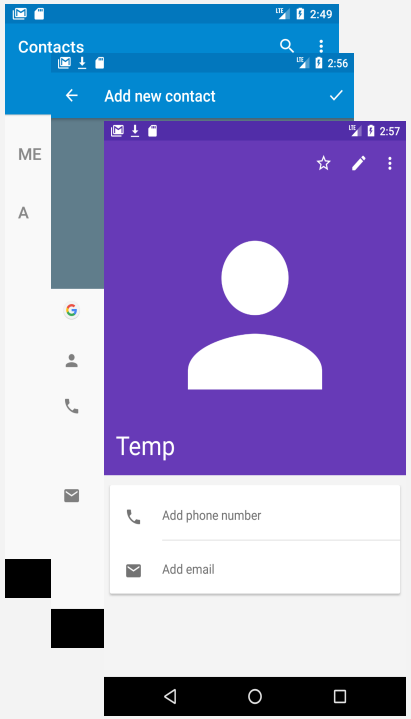
- Don’t abuse startActivity()!
  - As a solution: use finish() directly after starting an activity
    - finish() signals to the OS that the current activity is to be destroyed

Activity Stack

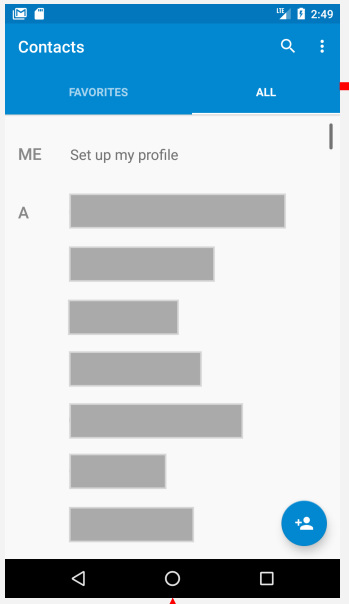


Let's look at another example…

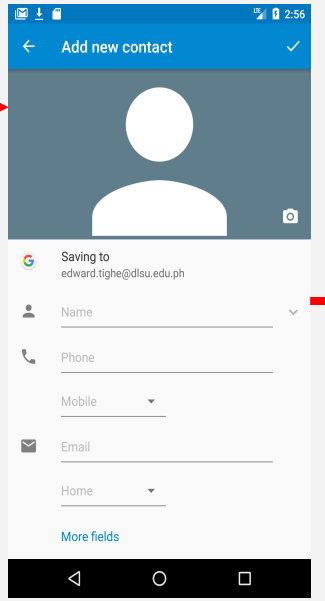
Activity Stack

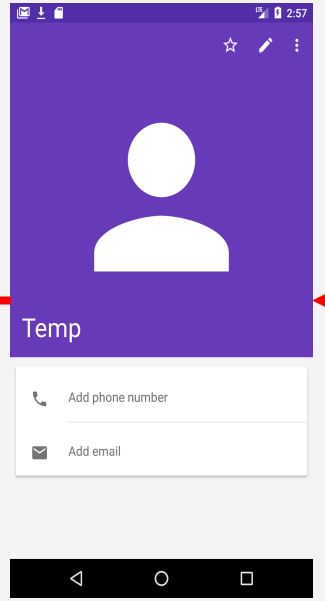


Basically, clicking the system's back button is very much like the finish() method



startActivity(i);





Uses the system's back button

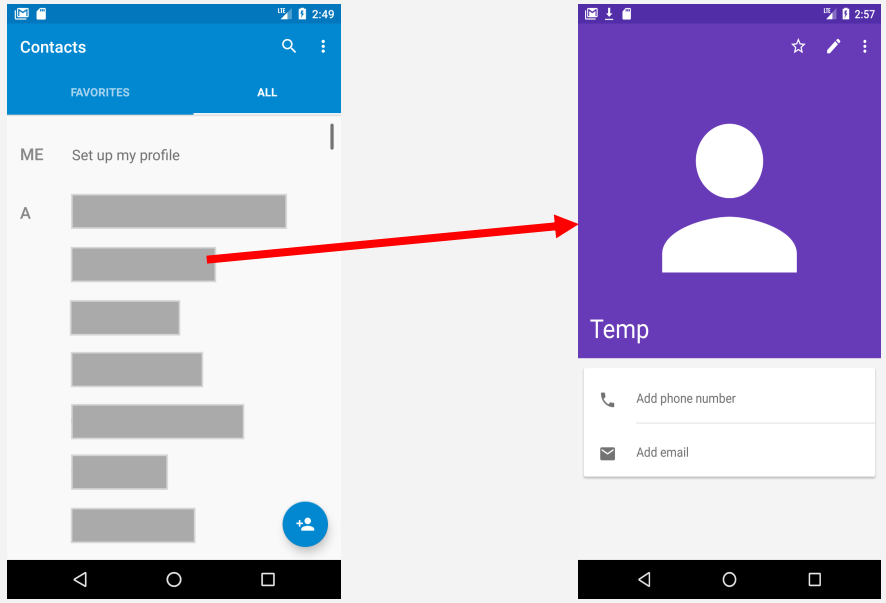
startActivity(i);
finish();

Moving from Activity to Activity

- Bottomline, be mindful of how you start and finish your activities
  - You might run into issues where the user would access data that's already supposed to be committed… which can possibly lead to more issues

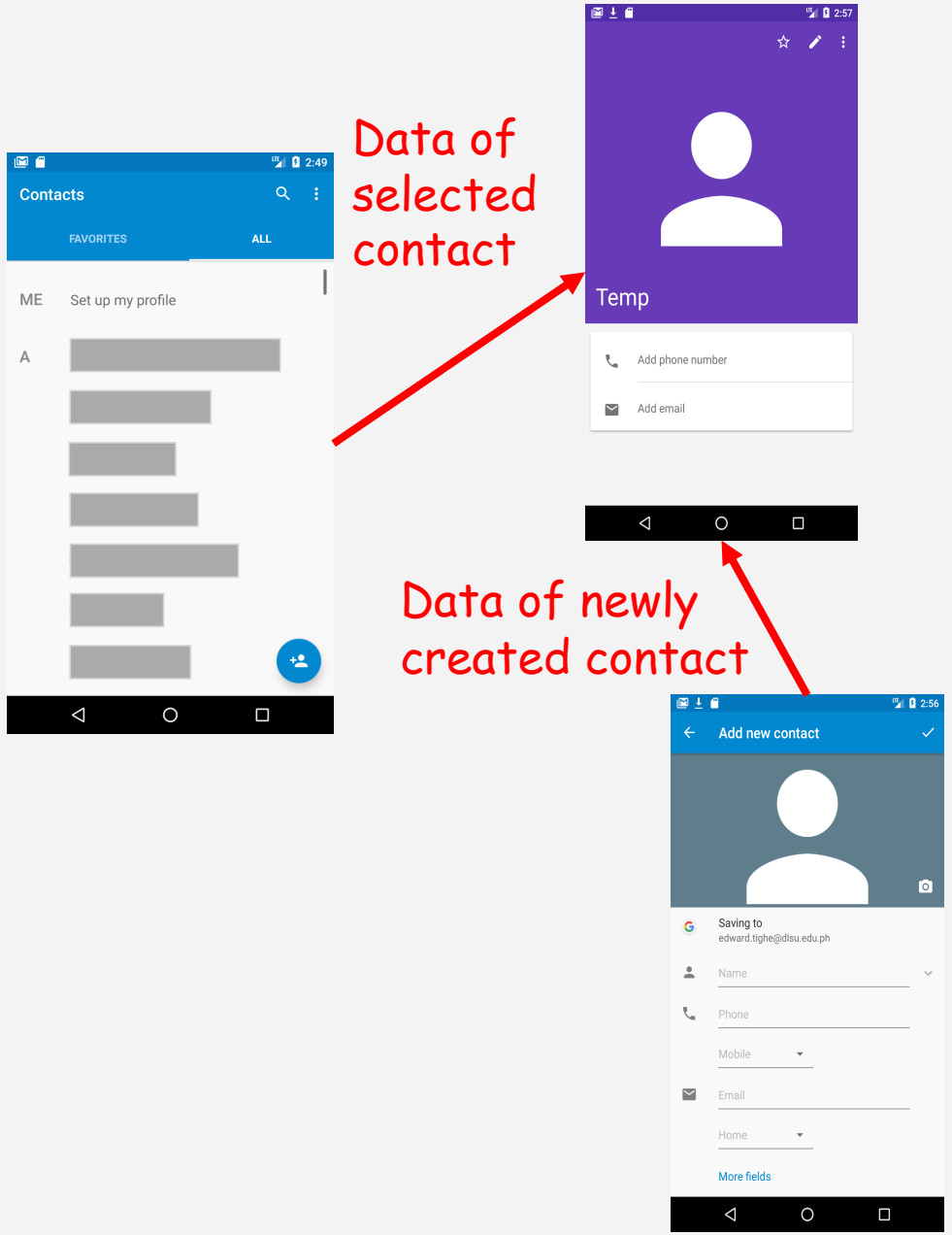
New Scenario…

- From the Home Screen of our Contacts App example, let's say we click on Contact #2…
  - Our app should open the View Contact Screen with Contact #2's data
  - Question: How do we get Contact #2's data from the Home Screen to the View Contact Screen?



Intents

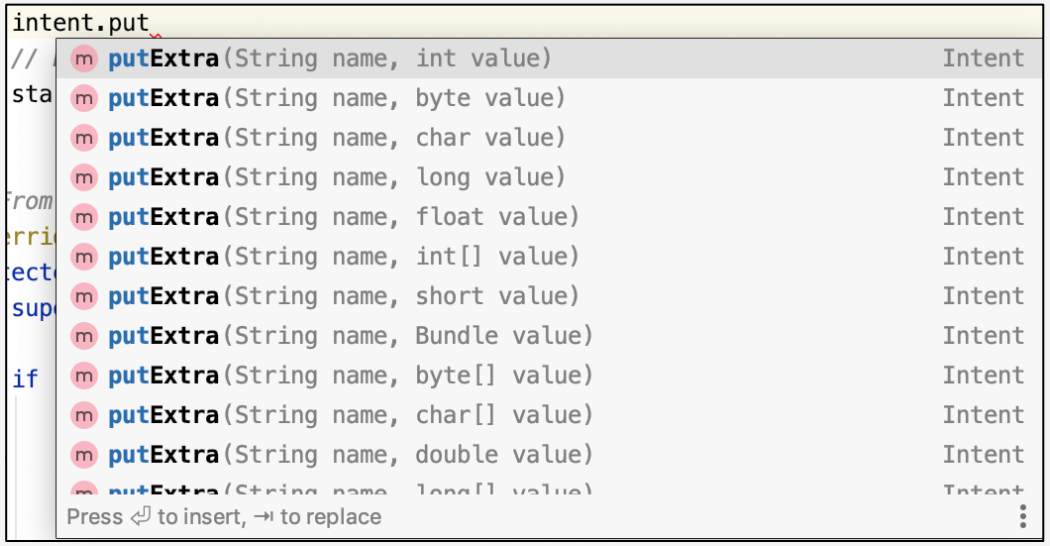
- Aside from communicating to the OS, Intents are also used to communicate between apps
- As Intents are like messages, we can store data inside of them an open them up in the destination
  - For this, we can use Extras!

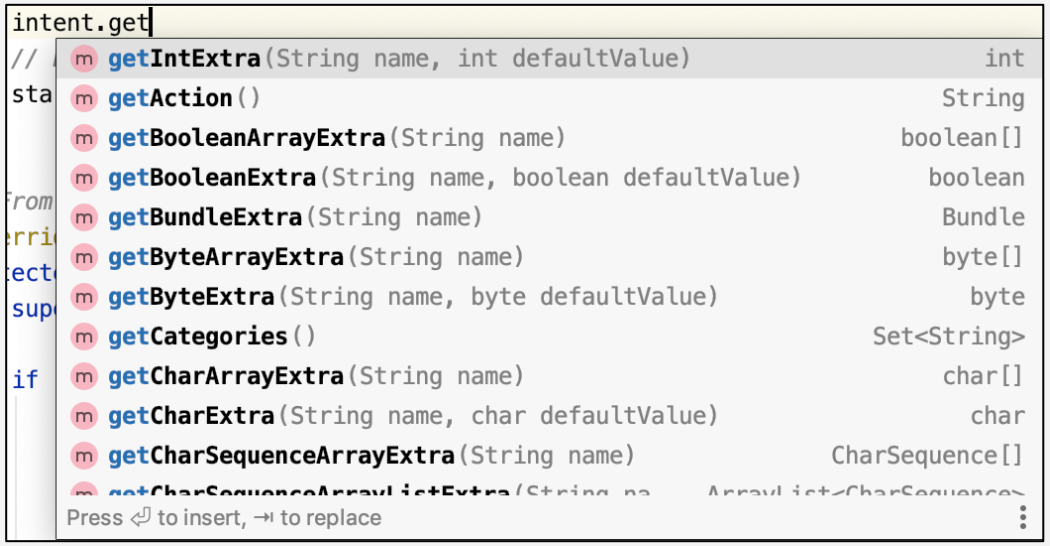


Intents and Extras

- To place data inside intents, utilize intent.putExtra()
  - This needs a String tag and a valu
- To retrieve data, utilize intent.get<type of data>Extra()
  - Refer to the API for the methods
  - Some methods require a default value

We also use getIntent() in the destination activity





Putting it All Together

In our example here, we want to send the integers 4 and 6 to the AdditionActivity, where we'll add both the numbers together…

Source Activity (MainActivity)

```
public static String ADDEND_1_TAG = "ADDEND_1";
public static String ADDEND_2_TAG = "ADDEND_2";

…

Intent intent = new Intent(
            MainActivity.this,
            AdditionActivity.class);
Intent.addExtra(ADDEND_1_TAG , 4)
Intent.addExtra(ADDEND_2_TAG , 6)
startActivity(intent);
```

From: MainActivity
To: AdditionActivity

ADDEND_1; 4
ADDEND_2; 6

Destination Activity (AdditionActivity)

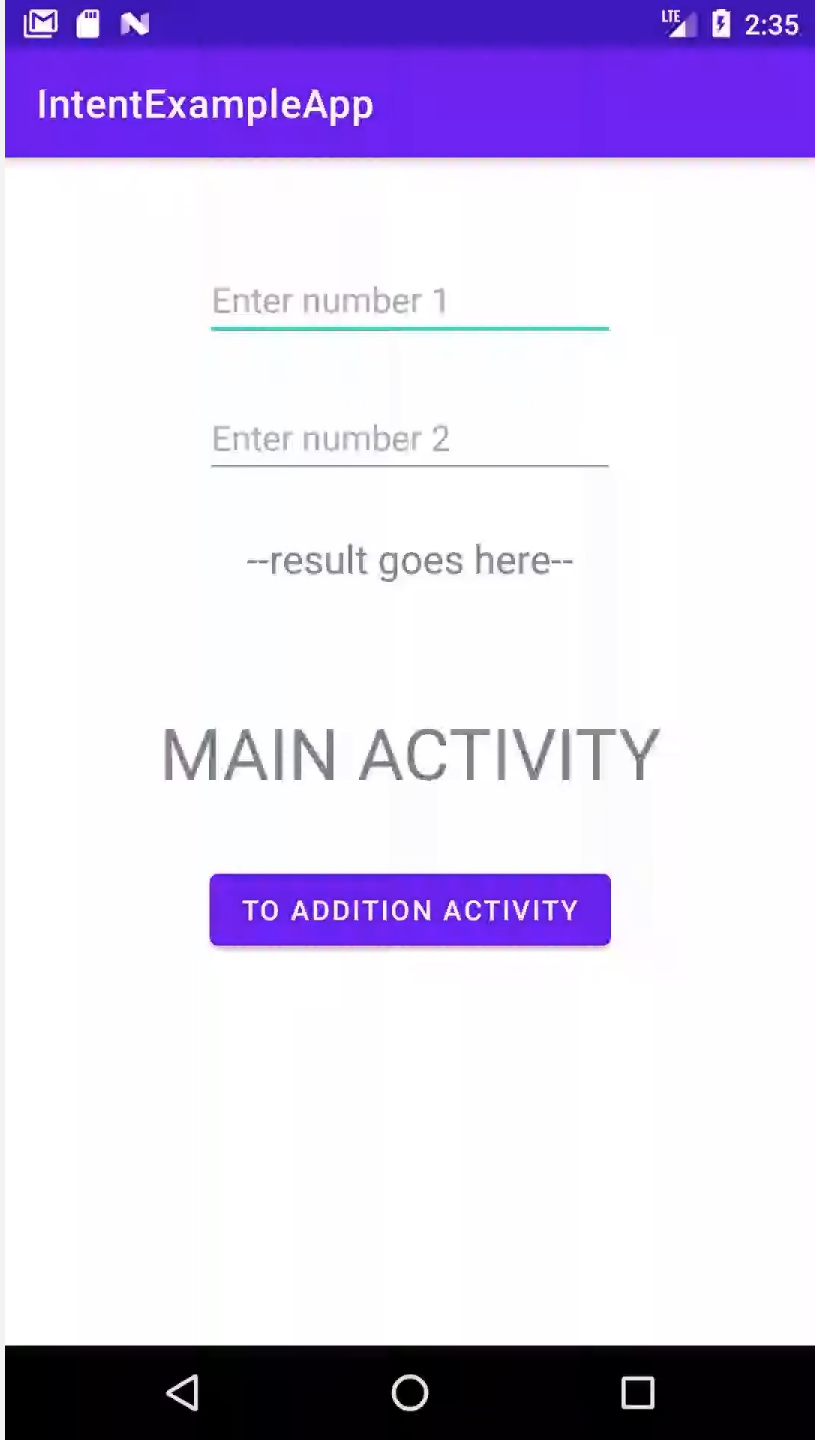
```
public static String ADDEND_1_TAG = "ADDEND_1";
public static String ADDEND_2_TAG = "ADDEND_2";

…

Intent intent = getIntent();
int addend1 = intent.getIntExtra(
            MainActivity.ADDEND_1_TAG,
            0); // Second argument is for a default value
int addend2 = intent.getIntExtra(
            MainActivity.ADDEND_2_TAG,
            0);
int result = addend1 + addend2
```

Expecting a Return Value(s)

- If we're expecting the destination activity to return a value to the source activity, we can make use of ActivityResultLauncher
  - Previously, the solution for this was to use startActivityForResult(), but this has since been depreciated

Expecting a Return Value(s)

- To demonstrate ActivityResultLauncher, let's consider the following application with 2 activities
  - First activity accepts 2 numbers and sends them to the second activity for adding
  - Second activity adds both numbers and sends the result back to the first activity for it to be displayed



```java
public class MainActivity extends AppCompatActivity {
    // Declaration of constants needed for Intent / ActivityResultLauncher
    public static String NUMBER1_KEY = "NUMBER1_KEY";
    public static String NUMBER2_KEY = "NUMBER2_KEY";

    private Button moveBtn;
    private EditText number1Etv, number2Etv;
    private TextView resultTv;

    // Code from: https://stackoverflow.com/questions/62671106/onactivityresult-method-is-deprecated-what-is
    private ActivityResultLauncher<Intent> myActivityResultLauncher = registerForActivityResult(
            new ActivityResultContracts.StartActivityForResult(),
            new ActivityResultCallback<ActivityResult>() {
                @Override
                public void onActivityResult(ActivityResult result) {
                    if(result.getResultCode() == Activity.RESULT_OK){
                        int res = result.getData().getIntExtra(AdditionActivity.RESULT_KEY,  defaultValue: 0);
                        resultTv.setText("RESULT: " + res);
                    } else if (result.getResultCode() == Activity.RESULT_CANCELED) {
                        resultTv.setText("RESULT: canceled");
                    }
                }
            });

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        initializeViews();
        this.moveBtn.setOnClickListener(view -> {
            // Create the Intent moving from MainActivity to AdditionActivity
            Intent intent = new Intent( packageContext: MainActivity.this, AdditionActivity.class);
            // Get the 2 numbers from EditTexts and insert to into the Intent object
            intent.putExtra(NUMBER1_KEY, Integer.parseInt(number1Etv.getText().toString()));
            intent.putExtra(NUMBER2_KEY, Integer.parseInt(number2Etv.getText().toString()));
            // Launch the Intent expecting a result
            myActivityResultLauncher.launch(intent);
        });
    }

    private void initializeViews() {...}
}
```

```java
public class AdditionActivity extends AppCompatActivity {
    // Declaration of constants needed for Intent / ActivityResultLauncher
    public static String RESULT_KEY = "RESULT_KEY";

    private Button moveBtn;
    private TextView logTv;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_addition);

        this.logTv = findViewById(R.id.logTv);
        this.moveBtn = findViewById(R.id.toMainActivity);

        Intent intent = getIntent();
        int number1 = intent.getIntExtra(MainActivity.NUMBER1_KEY,  defaultValue: 0);
        int number2 = intent.getIntExtra(MainActivity.NUMBER2_KEY,  defaultValue: 0);
        int results = number1 + number2;

        this.logTv.setText("LOG" + "\nNumber 1: " + number1 + "\nNumber 2: " + numbe

        this.moveBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent return_intent = new Intent();
                return_intent.putExtra(RESULT_KEY, results);
                setResult(Activity.RESULT_OK, return_intent);
                finish();
            }
        });
    }
}
```

1 → Intent intent = new Intent(packageContext: MainActivity.this, AdditionActivity.class);

2 → intent.putExtra(NUMBER1_KEY, Integer.parseInt(number1Etv.getText().toString()));
intent.putExtra(NUMBER2_KEY, Integer.parseInt(number2Etv.getText().toString()));

3 → myActivityResultLauncher.launch(intent);

4 → public void onCreate(Bundle savedInstanceState) {

5 → int number1 = intent.getIntExtra(MainActivity.NUMBER1_KEY, defaultValue: 0);
int number2 = intent.getIntExtra(MainActivity.NUMBER2_KEY, defaultValue: 0);
int results = number1 + number2;

6 → this.logTv.setText("LOG" + "\nNumber 1: " + number1 + "\nNumber 2: " + numbe

7 → Intent return_intent = new Intent();
return_intent.putExtra(RESULT_KEY, results);

8 → setResult(Activity.RESULT_OK, return_intent);
finish();

9 → public void onActivityResult(ActivityResult result) {

10 → if(result.getResultCode() == Activity.RESULT_OK){

11 → int res = result.getData().getIntExtra(AdditionActivity.RESULT_KEY, defaultValue: 0);
resultTv.setText("RESULT: " + res);

At this point, some of you might be thinking this is a lot of effort to move data around... and you're right to think so. However, given how Android treats Activities, this is a way to transfer data around

Questions?

Summary

- Activities are basic components of any Android application
  - Streamline functionalities of an application
    - E.g. Home Page, Settings, View, Add/Edit, etc.
- Context allows access to resources and classes
  - Is needed by certain objects, like Views
  - Can be given at different levels: Activity, Application

Summary

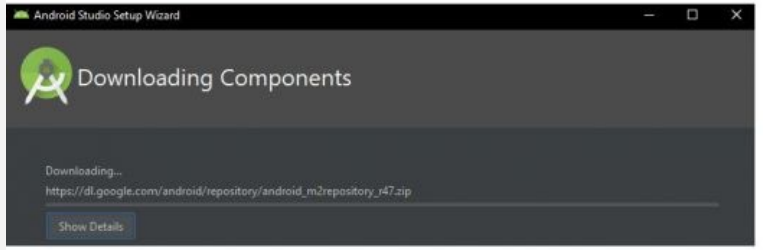
- Its rare for an application to have only one Activity
- To move to another Activity, we need to create an intent and use startActivity(i)
  - We have to be mindful of how our Activities can stack and should use finish() when needed
- We can also send data to other activities through Extras
- If we're expecting the destination to return a result, we can use ActivityResultLauncher

For access to the project for the ActivityResultLauncher code, please see the Module in Canvas

Thanks everyone!

Me:

Android Studio:



Me:

Image sauce: https://devrant.com/rants/1897105/making-a-meme-to-pass-the-time-all-this-waiting-only-because-i-wanted-to-try-one