Android App Development

While Android App Development in Java may not necessarily be the best choice, it does however provide a foundation for future app development in other languages and for other platforms. By the end of this set of notes, you would have (hopefully) created an android application!

Overview

1. Installation
2. Introduction
3. Creating an application
4. Layouts
5. Updating our application
6. Intents
7. Activity Lifecycle
8. Final update to our application

Installation

Windows

1. Download the latest version of android studio → https://developer.android.com/studio?hl=fr (be sure to download the .exe)
2. Double Click to launch it
3. Follow the setup wizard and install any SDK (Software Development Kit) packages it recommends.

For more details: https://developer.android.com/studio/install?hl=fr#windows

Mac

1. Download the latest version of android studio → https://developer.android.com/studio?hl=fr (be sure to download the .dmg)
2. Launch the Android Studio DMG file
3. Drag and drop Android Studio into the Applications folder, then launch Android Studio.
4. Select whether you want to import previous Android Studio settings, then click **OK**. (**You should pick no if you have never installed Android Studio before**)
5. The Android Studio Setup Wizard guides you through the rest of the setup, which includes downloading Android SDK components that are required for development.

For more details: https://developer.android.com/studio/install?hl=fr#mac

Linux

If you use Linux, or any Unix-like system, you should be able to figure it out on your own.

The link to the Linux installation guide is here → https://developer.android.com/studio/install?hl=fr#linux

Introduction to Android App Development

Android is a mobile system maintained by Google. Android provides libraries for many system features such as contacts, phone dialing, notifications, graphics, database, etc. For this entire App Development notes, we will be sticking to **Android 12 (API 31)**.

Project Structure

Important files

- `AndroidManifest.xml`
  - Overall Project configuration and settings
- `build.gradle`
  - Main build config file
  - Gradle is a build and compile management system

Important directories

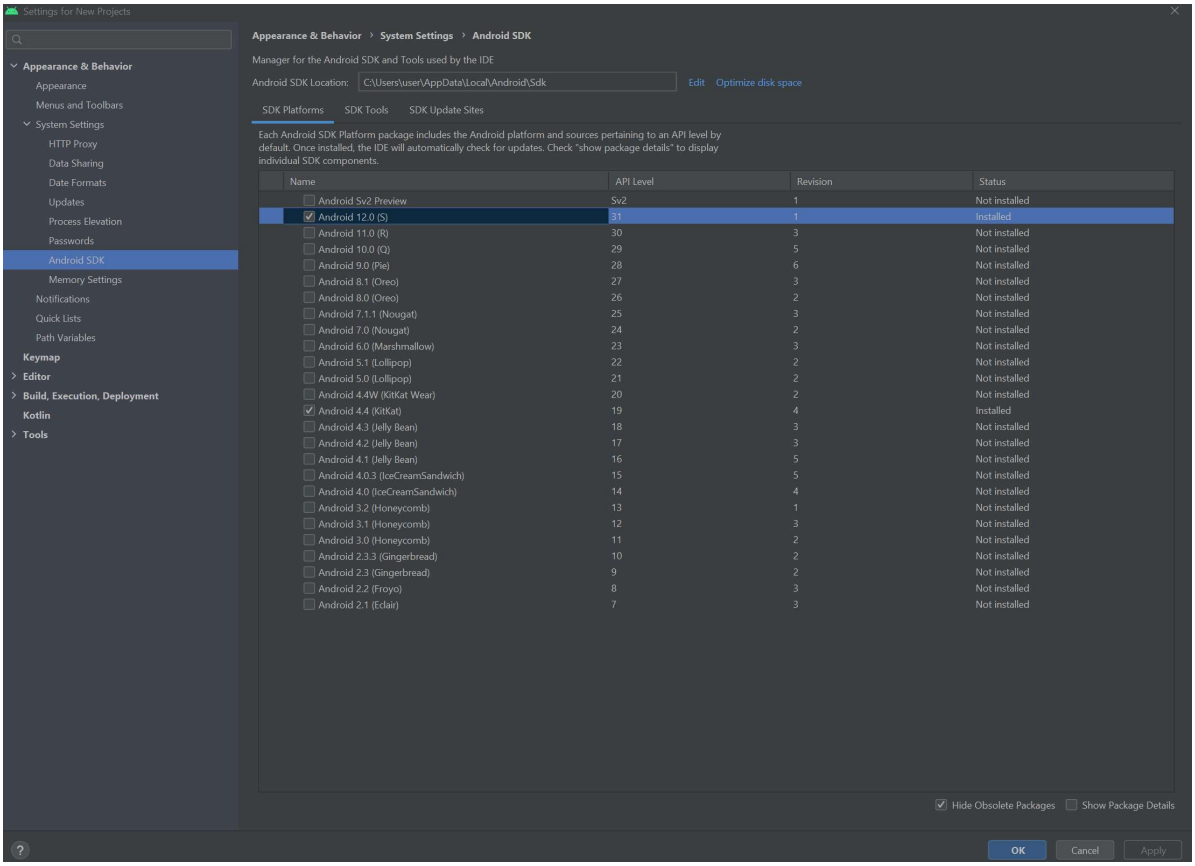
- `src/java/...`
  - Source code for your Java classes
- `res/...` → Resource Files (mostly XML files)
  - `drawable/` → Images
  - `layout/` → Descriptions of GUI layout
  - `menu/` → Overall app menu options
  - `values/` → Constant Values and Arrays
  - `strings` → Localization Data
  - `styles` → General Appearance Styling

Android Terminology

Term	Definition
Activity	A single screen of UI that appears in your app. Fundamental unit of GUI in Android Applications.
View	Items that appear onscreen in an activity
Widget	GUI control such as buttons or text fields
Layout	Invisible container that manages positions/sizes of widgets
Event	Action that occurs when a user interacts with widgets
Action Bar	Meny of common action at the top of the application
Notification Area	Topmost system menu and icons

Creating an Application

For this section, please ensure you currently have android studio opened and you are at the front page.

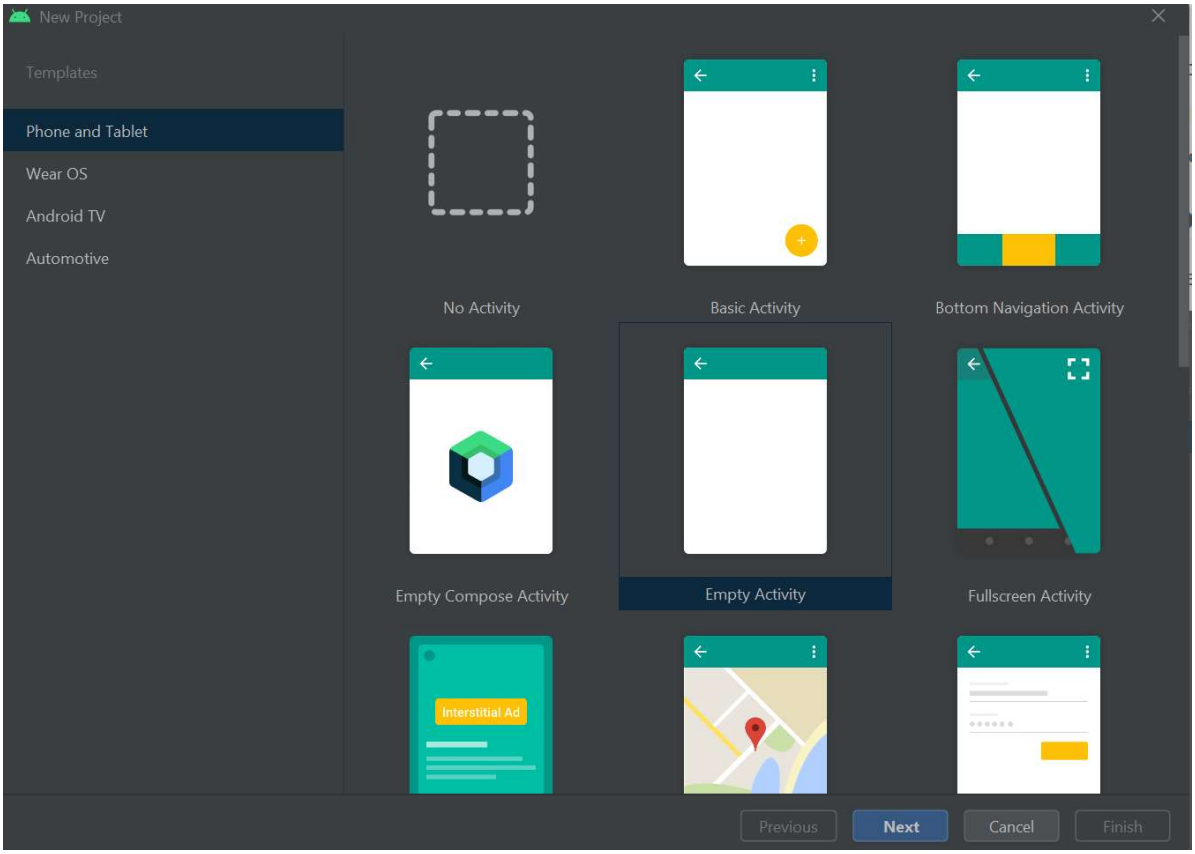


It should look something like the above.

Installing Android 12 SDK

If for some reason, it is not already installed, follow the steps below. (It should have been installed when installing Android Studio)

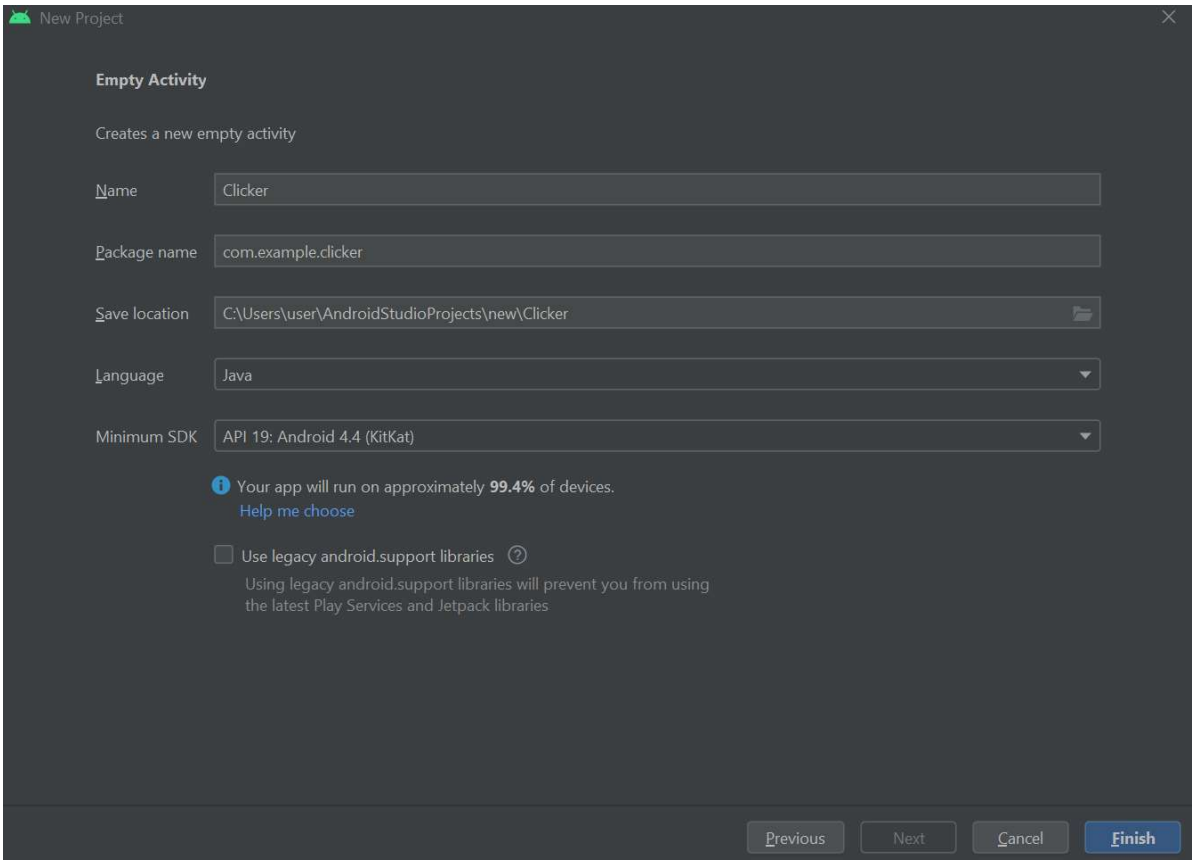
1. Click on `More Actions`
2. Click on SDK Manager



3. Ensure that Android 12 (API level 31) is clicked. (If it is already clicked, skip to the next section).
4. If you have just clicked the checkbox for Android 12, Click `Apply`, then click `Ok`. Wait for it to finish downloading and installing. After that, Click `Finish`, and you can close this window to return back to the main screen.

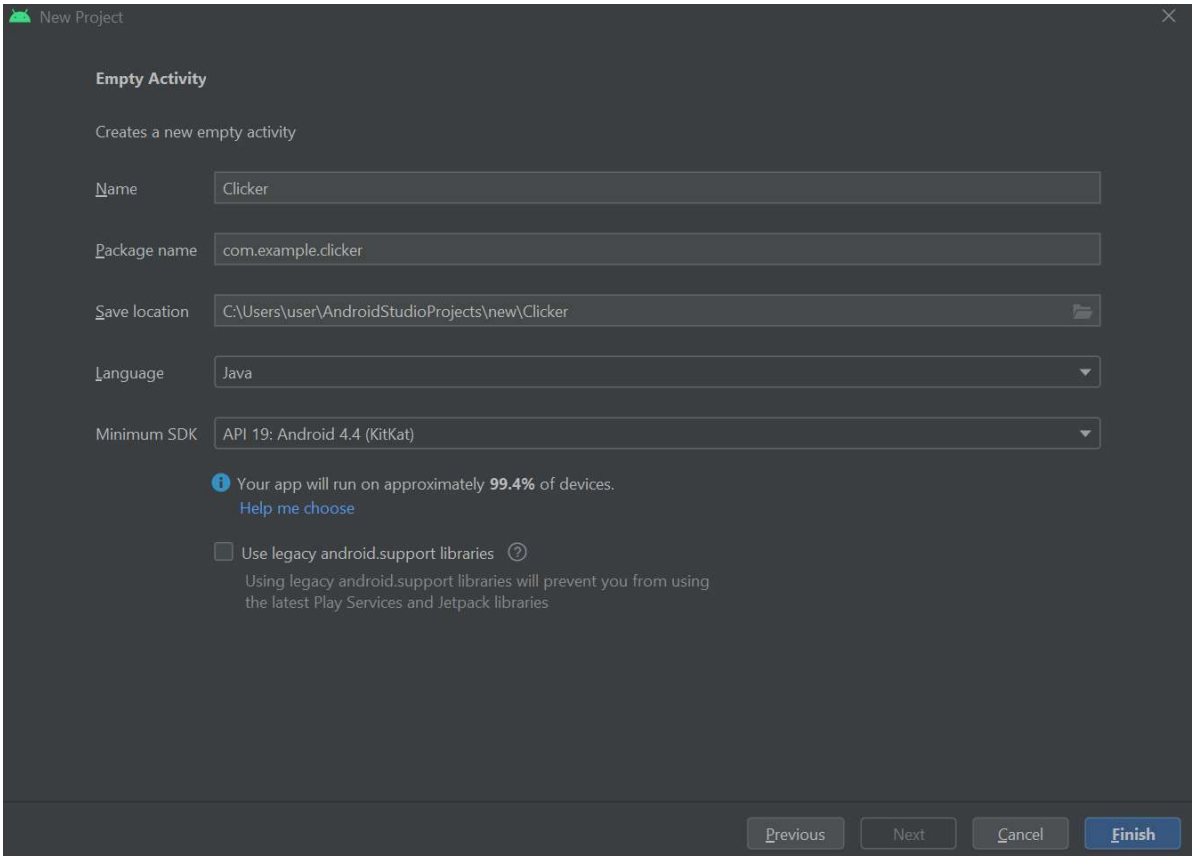
Creating a New Project

1. Click the "Add" button to create a New Project
2. Select Empty Activity

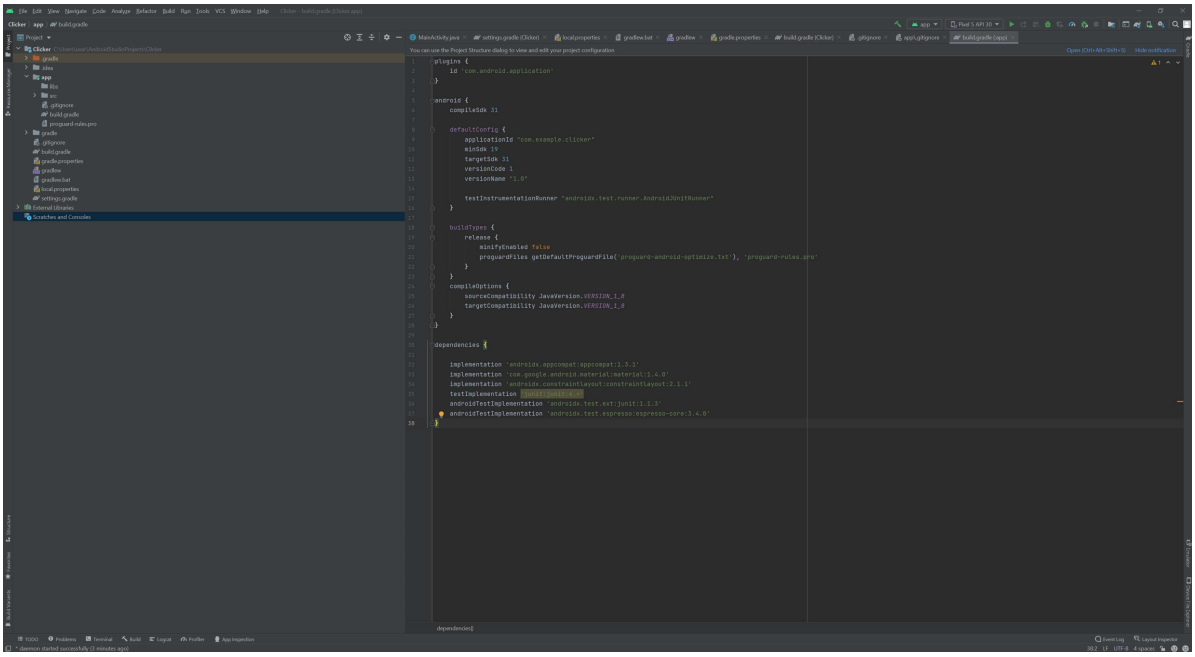


3. Click Next
4. Configure the Application
    1. Name = `Clicker`
    2. package name = `com.example.clicker`
    3. Save Location: Wherever you plan to save your application
    4. **Language =** `Java`
    5. **Minimum SDK =** `API 19: Android 4.4 (KitKat)`

    It should look like the picture below.



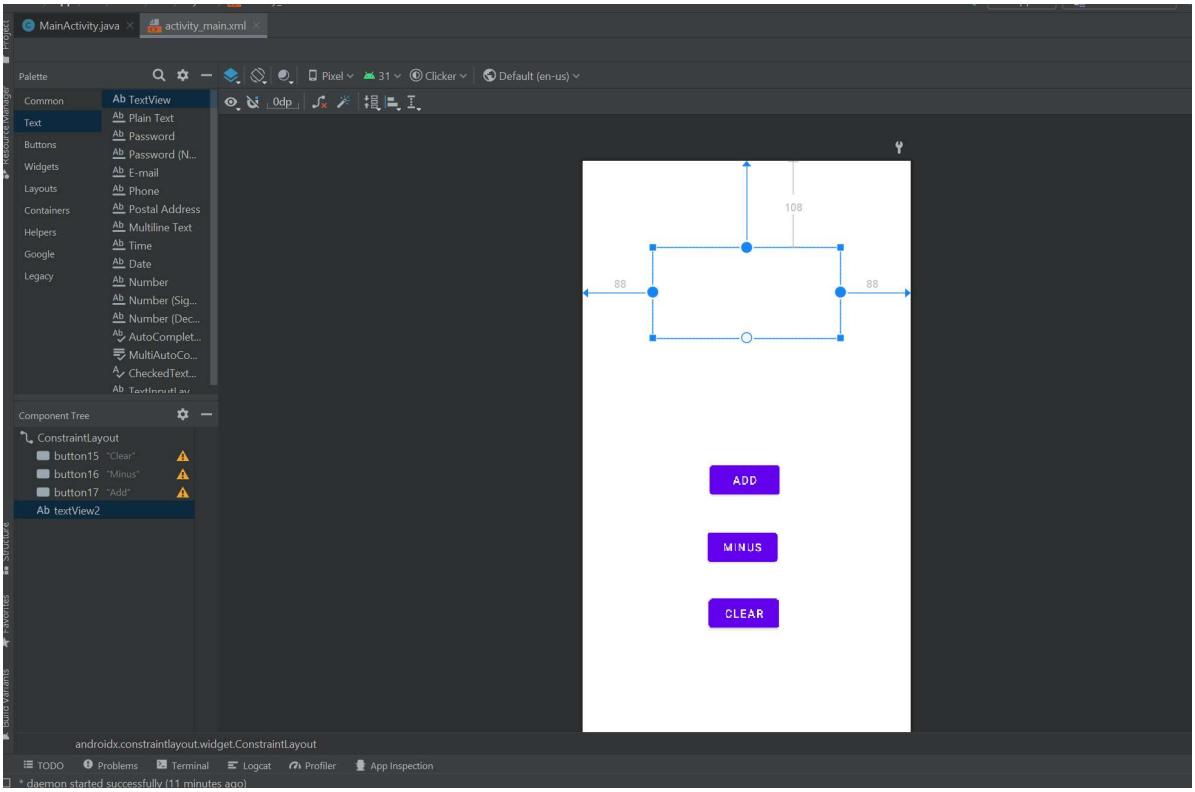
5. Click Finish
6. (Let Android Studio load and index files)
7. You should be greeted by the following screen **(or something similar)**



Designing a User Interface

- Open the XML file for your layout (It should be `activity_main.xml` for the default layout)
- Drag Widgets from the left **Palette** to the preview image
- Set their properties in the lower-right Common Attributes Panel

We will be adding **3 buttons and 1 TextView Widget**. You should end up with something similar to this.

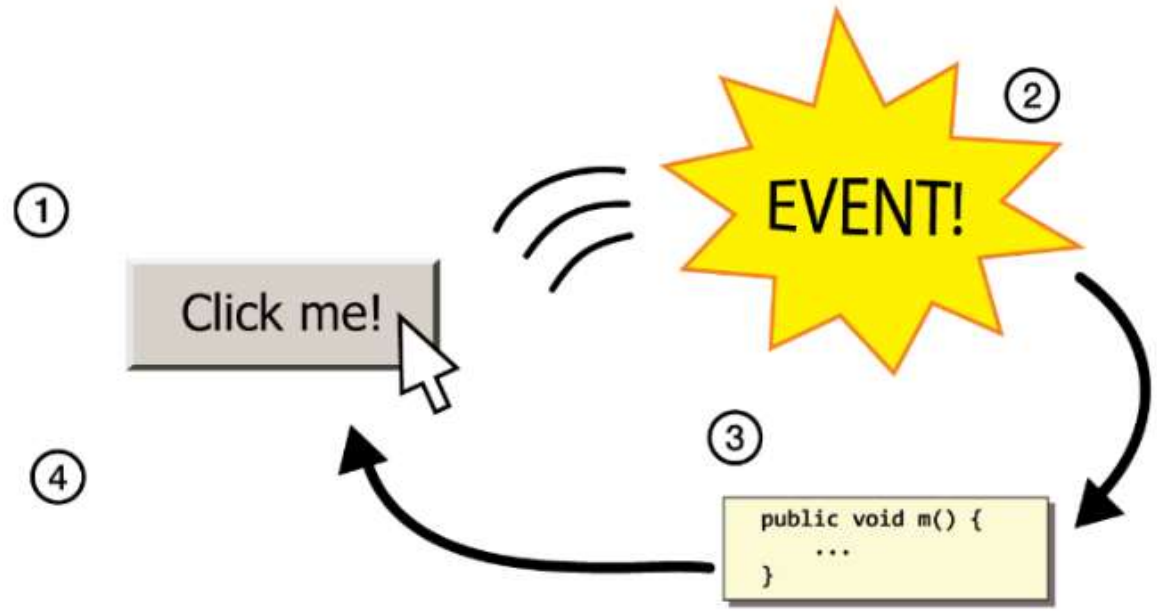


Note, your `button` Number and `textView` Number may be different, but it should be fine. `textView` is intentionally left blank for our application.

If you encounter any errors regarding constraints, use the GUI to set the constraints of the widgets. This can be found at the top of the properties panel under Layout.

Events (Overview)

What is an event? An event is simply an external stimulus your program can respond to. Common events include mouse motion, tapping, keys pressed etc. In most graphical programs, event-driven programming is used. This means the execution of your program is largely dictated by user events.



As such, to respond to events in a program, we need to:

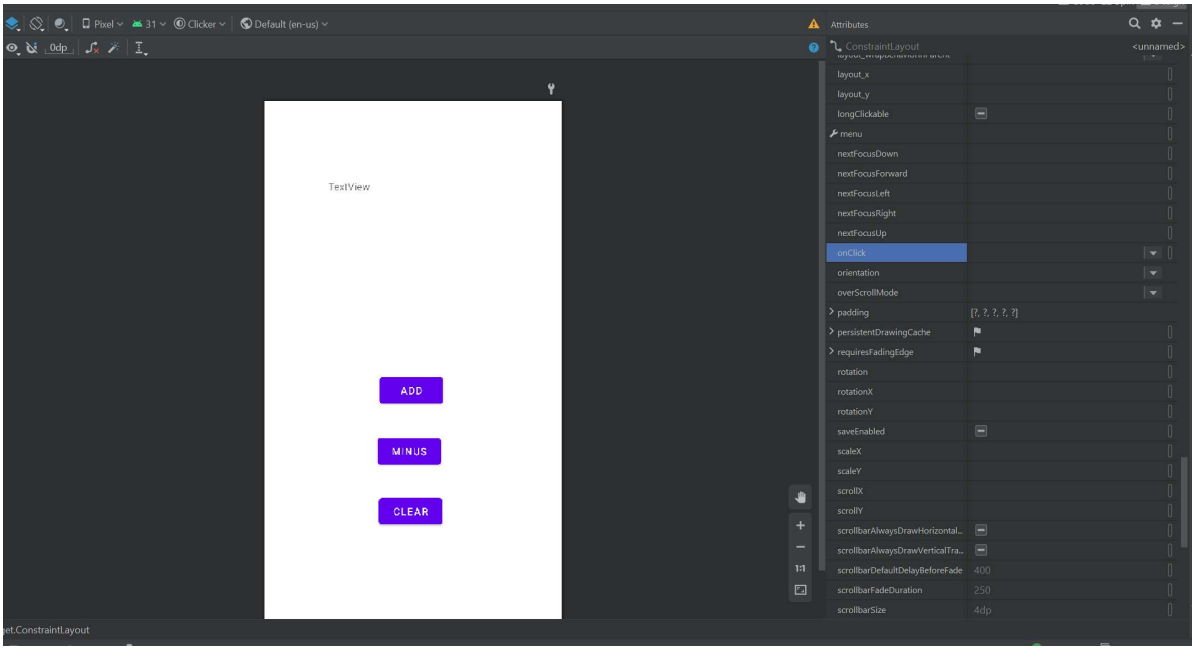
1. Write methods to handle each event ("listener" methods)

2. Attach those methods to GUI widgets (like your buttons)

Programming our application

First, let's set an event listener for our buttons.

1. Select the widget (the button) in the Design view
2. scroll down its properties until you find **onClick**
3. type the name of the method you'll want to handle that click



For now, we will assign the follow method names to each button

ADD: `buttonAdd_click`
MINUS: `buttonMinus_click`
CLEAR: `buttonClear_click`

In addition, TextView should have **gravity** set to `center`, and all buttons should have **clickable** set to `true` (ticked). Your XML should look something like this:

```
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="163dp"
        android:layout_marginTop="504dp"
        android:layout_marginEnd="160dp"
        android:clickable="true"
        android:onClick="buttonClear_click"
        android:text="Clear"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="163dp"
```

```
        android:layout_marginTop="340dp"
        android:layout_marginEnd="160dp"
        android:clickable="true"
        android:onClick="buttonAdd_click"
        android:text="Add"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="163dp"
        android:layout_marginTop="416dp"
        android:layout_marginEnd="160dp"
        android:clickable="true"
        android:onClick="buttonMinus_click"
        android:text="Minus"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="235dp"
        android:layout_height="114dp"
        android:layout_marginStart="88dp"
        android:layout_marginTop="110dp"
        android:layout_marginEnd="88dp"
        android:gravity="center"
        android:text="TextView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

Now, we will move on to programming the event listener. Heading over to MainActivity.java, we will first declare our event listener methods. Your code should look something like this

```
//truncated import statements
public class MainActivity extends AppCompatActivity {

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

    public void buttonClear_click(View view) {
        //your code goes here
    }

    public void buttonMinus_click(View view) {
        //your code goes here
```

```
    }

    public void buttonAdd_click(View view) {
        //your code goes here
    }

}
```

View Objects

Let us talk briefly about View Objects. Each widget has an associated Java Object you can access. They are all subclasses of the parent class `View`. View objects have many get and set methods that correspond to the properties in the Design view. For example, background, bottom, margin, etc etc

For example, a Button's text property has the following methods

```
public String getText()
public void setText(String text)
```

For more details on these methods available to widgets, you can refer to the Android Developer reference. https://developer.android.com/reference

Interacting With Widgets

Now how do we interact with widgets?

1. In Design view, we give that view a unique ID property value (this should be done by default)
2. In Java code, we call `findViewById()` to access its View Object. For example,

```
public void button_onClick(View view){
    TextView tv = (TextView) findViewById(R.i.mytextview);
    tv.setText("You clicked it!")
}
```

Programming our App

Going back to our application, we want to program a clicker application. As such, we will need to,

1. Declare a variable to hold the number of clicks
2. Update this variable every time a button is clicked
3. Update the TextView of the application every time the button is clicked

Your code should end up looking something like this

```
package com.example.clicker;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    TextView text;
    int count = 0;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        text = (TextView) findViewById(R.id.textView);
        text.setText(String.valueOf(count));
    }

    public void buttonClear_click(View view) {
        count = 0;
        text.setText(String.valueOf(count));
    }
    public void buttonMinus_click(View view) {
        count -= 1;
        text.setText(String.valueOf(count));
    }

    public void buttonAdd_click(View view) {
        count += 1;
        text.setText(String.valueOf(count));
    }
}
```

Toast Messages

A "Toast" is a pop-up message that appears for a short time. Toasts are useful for displaying short updates in response to events.

The syntax is as follows

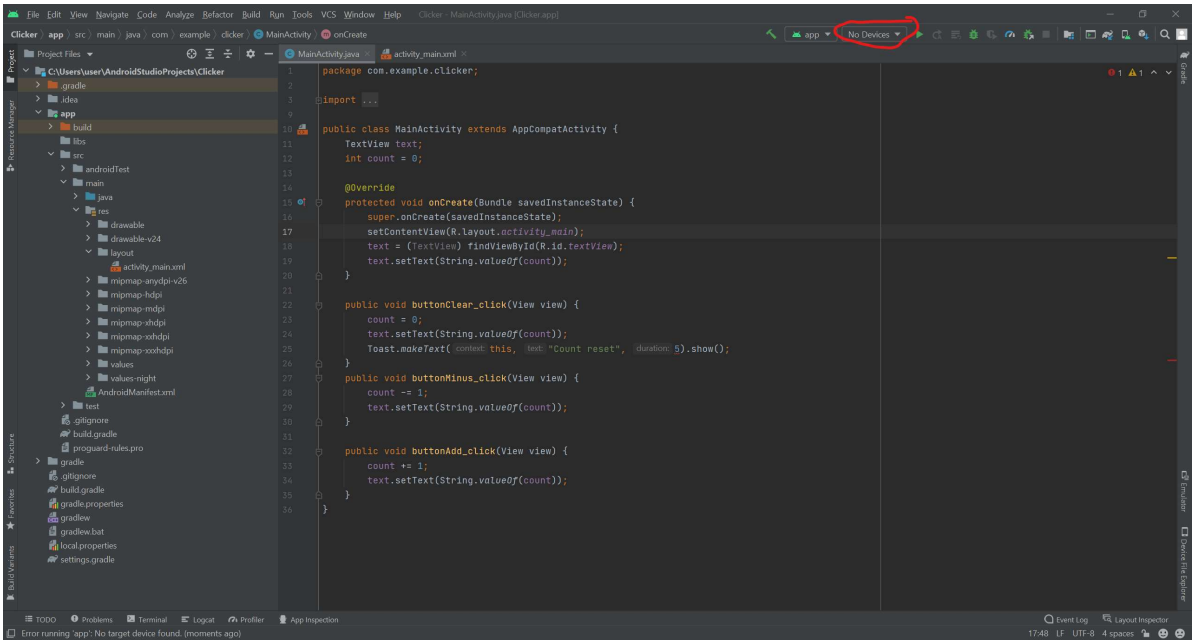
```
Toast.makeText(this, "message", duration).show();
```

When the clear button is pressed, we will create a new Toast to let the user know the count has been reset. Your code should look something like this

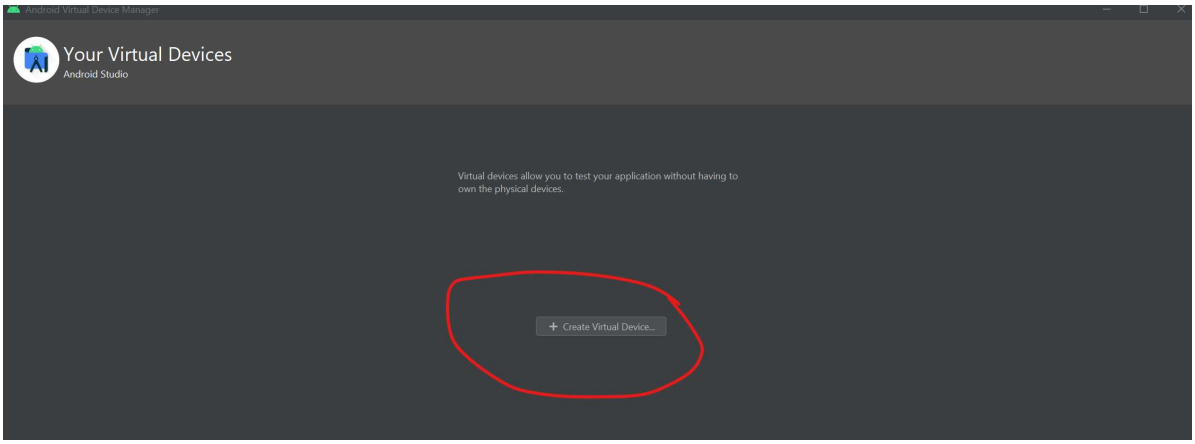
```
public void buttonClear_click(View view) {
        count = 0;
        text.setText(String.valueOf(count));
        Toast.makeText(this, "Count reset", Toast.LENGTH_LONG).show();
    }
```

Installing Android Virtual Device and Running Application

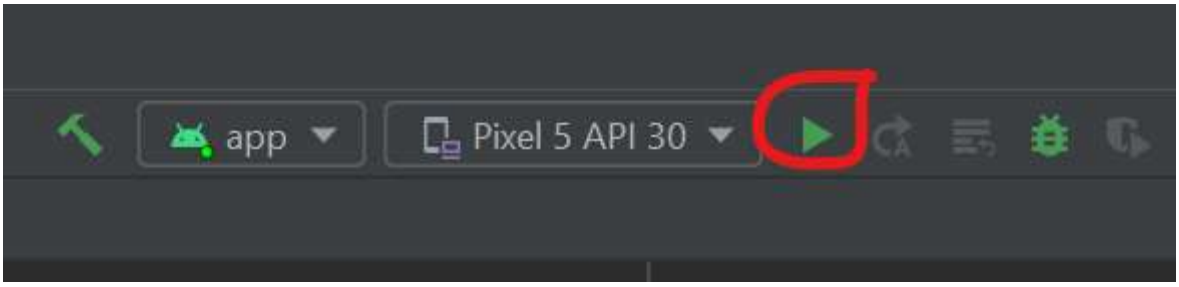
With the code complete, we are now ready to run our application. To do this, we will install an Android Virtual Device to run our application. Your screen should look something like this



1. Click on No Devices, then click on AVD Manager
2. Click Create Virtual Device



3. Choose Pixel 5, Click Next, Select Android 11 (Release name R), Click Next, and Click Finish
4. Close the Virtual Device Screen, and click on the Run button. (The Green Play button)



5. Wait for the Virtual Device to load and Install your application.

Congratulations, you have just created your first application!

Layouts

Overview

So how does one specify where a component appears? In Java, we make use of Layout managers.

Layout managers:

- Decide where to position each component based on certain rules or criteria
    - Example: placing four buttons in a 2x2 grid
- Are flexible and general. This allows it to work better with a variety of devices

In Android, layouts are described in XML. XML is a:

- language for describing hierarchical text data.
- uses tags that contains elements and atrributes. Note, tags can be nested
- Some tags are opened and closed, otherss self close.
- is **case sensitive**

```
<!-- This is a comment -->
<!-- Open/Closed -->
<element attr="value" attr="value"> ... </element>

<!-- Self Closing -->
<element attr="value" attr="value" />
```

To change layouts, go to the **Text** view for your layout XML file. You can modify the opening and closing tags to the new layout type, such as LinearLayout. You can head back to the **Design** view and add widgets as you like.

Linear Layout

A linear layout:

- lays out widgets/views in a single line
- has either a horizontal (default) or vertical orientation

- items do not wrap around if they reach edge of screen

Example of Linear Layout:

```
<LinearLayout ...
        android:orientation="horizontal"
        tools:context=".MainActivity">
    <Button ... android:text="Button 1" />
    <Button ... android:text="Button 2 Hooray" />
    <Button ... android:text="Button 3" />
    <Button ... android:text="Button 4
                        Very Long Text" />
</LinearLayout>
```

- In our examples, we'll use `...` when omitting boilerplate code that is auto-generated by Android Studio and not relevant to the specific example at hand.

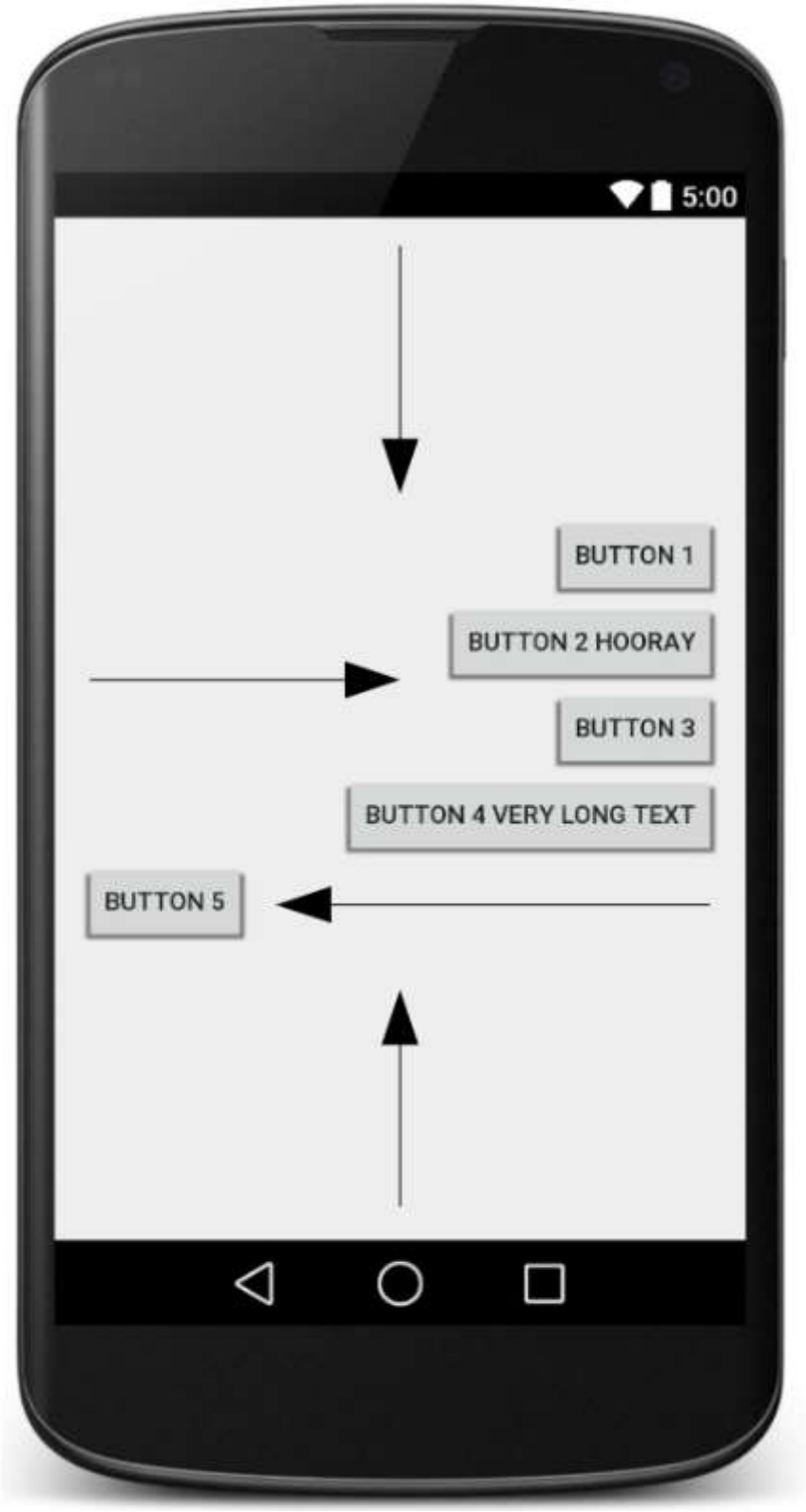
Gravity

Gravity is the alignment direction that widgets are pulled. Gravity:

- can either be top, bottom, left, right or center
- can combine multiple with `|`
- `gravity` sets all widgets, while `layout_gravity` sets an individual widget

Example:

```
<LinearLayout ...
    android:orientation="vertical"
    android:gravity="center|right"
    <Button ... android:text = "Button 1" />
    <Button ... android:text = "Button 2 Hooray" />
    <Button ... android:text = "Button 3" />
    <Button ... android:text = "Button 4 Very Long Text" />
    <Button ... android:text = "Button 5" android:layout_gravity = "left" />
</LinearLayout>
```



Weight

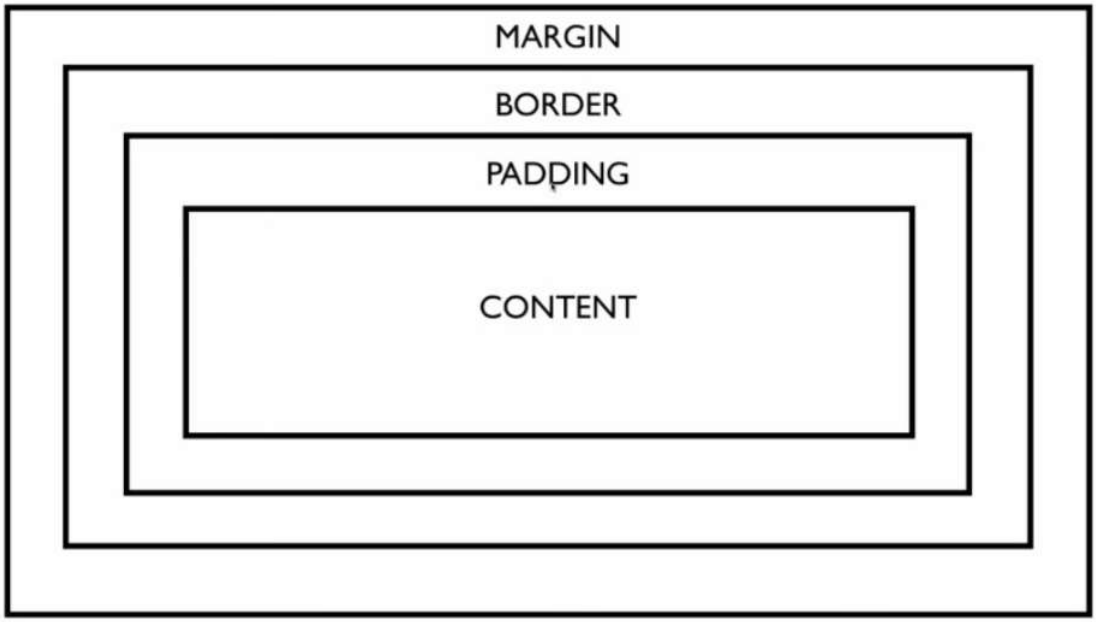
Weight defines how much space a view will consume compared to other views within a LinearLayout. A widget with weight **K** gets **K/total** of total size.

Example:

```
<LinearLayout ...
        android:orientation="vertical">
    <Button ... android:text="B1"
              android:layout_weight="1" />
    <Button ... android:text="B2"
              android:layout_weight="3" />
    <Button ... android:text="B3"
              android:layout_weight="1" />
</LinearLayout>
```

Widget Box Model

- **Content**: every widget or view has a size (width * height) for its content
- **Padding**: Can artifically increase a widget's size by applying padding just outside its content
- **Border**: A line around the edge of a widget
- **Margin**: Seperation from neightbouring widgets on screen



Sizing Individual Widgets

One can size individual widgets with width and height. The width and height of a widget can be:

1. `wrap_content`: exactly large enough to fit the widget's content
2. `match_parent`: as wide or tall as 100% of the screen or layout
3. A specific fixed width such as device pixels (`dp`), device-independent pixels (`dip`) or scaling pxels (`sp`)

Example:

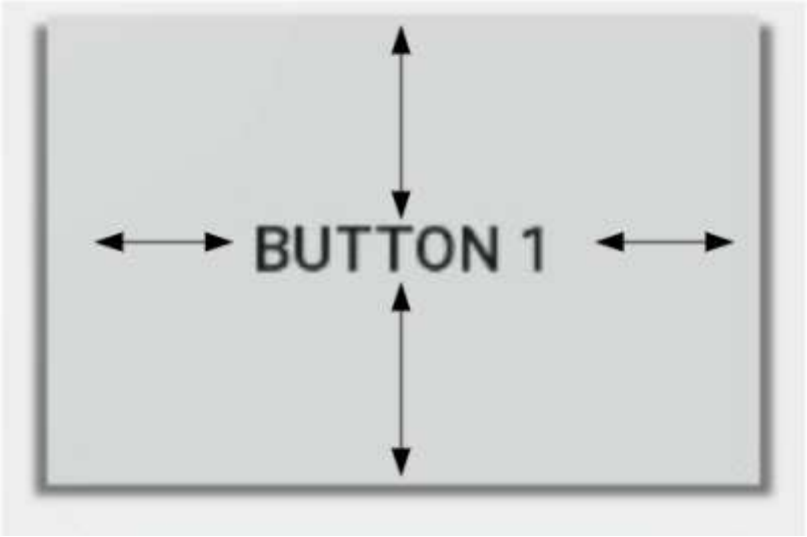
```
<Button ...
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content" />
```

Padding

Padding adds extra space inside widgets. One can set padding to adjust all sides (`paddingTop`, `Bottom`, `Left` or `Right`). Padding is usually set to specific values such as 10dp.

Example:

```
<LinearLayout ...
              android:orientation = "vertical">
    <Button ... android:text = "Button 1" android:padding = "50dp" />
</LinearLayout>
```



Margin

Margin is the extra space outside a widget to seperate it from others.

- `layout_margin` adjusts all sides. Such as `layout_marginTop`, `Bottom`, `Left` or `Right`
- Is usually set to specific values such as 10dp

Example:

```
<LinearLayout ...
              android:orientation = "vertical">
    <Button ... android:text = "Button 1" android:layout_margin = "50dp" />
</LinearLayout>
```

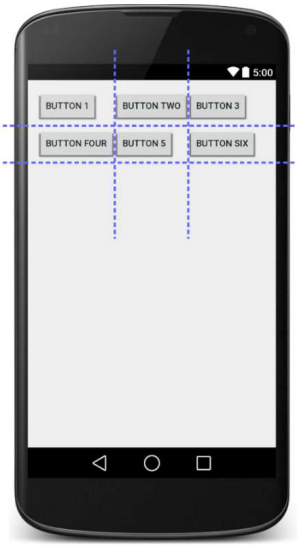
Grid Layout

A grid layout:

- Lays out widgets/views in lines of rows and columns
    - orientation attribute defines row-major or column-major order
- By default, rows and columns are equal in size
    - each widget is placed into "next" available row/column index unless given an explicit `layout_row` and `layout_column` attribute

Example:

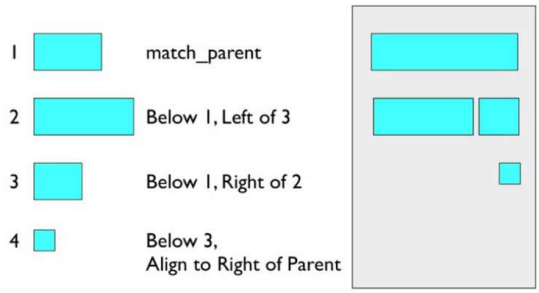
```
<GridLayout ...
        android:rowCount="2"
        android:columnCount="3"
        tools:context=".MainActivity">
    <Button ... android:text="Button 1" />
    <Button ... android:text="Button Two" />
    <Button ... android:text="Button 3" />
    <Button ... android:text="Button Four" />
    <Button ... android:text="Button 5" />
    <Button ... android:text="Button Six" />
</GridLayout>
```



Relative Layout

In a relative layout, each widget position and size are relative to other views.

- relative to "parent" (the activity itself)
- relative to other widgets/views
- x-positions of reference (left, right, center)
- y-positions of reference (top, bottom, center)



Properties for x,y relative to another widget:

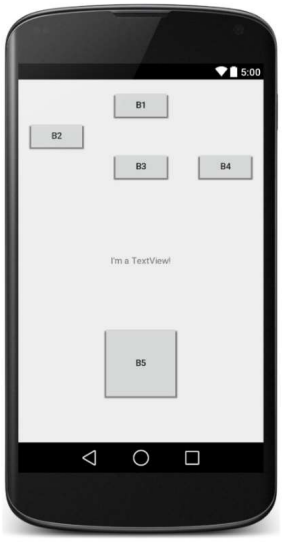
- `layout_below`, `above`, `toLeftOf`, `toRightOf`
  - set these to the ID of another widget in the format `"@id/theID"`

Properties for x,y relative to layout container (the activity):

- `layout_alignParentTop`, `Bottom`, `Left`, `Right`
  - set these flags to a boolean value of "true" to enable them
- `layout_centerHorizontal`, `Vertical`, `InParent`
  - set these flags to "true" to center the control within its parent in a dimension

Example:

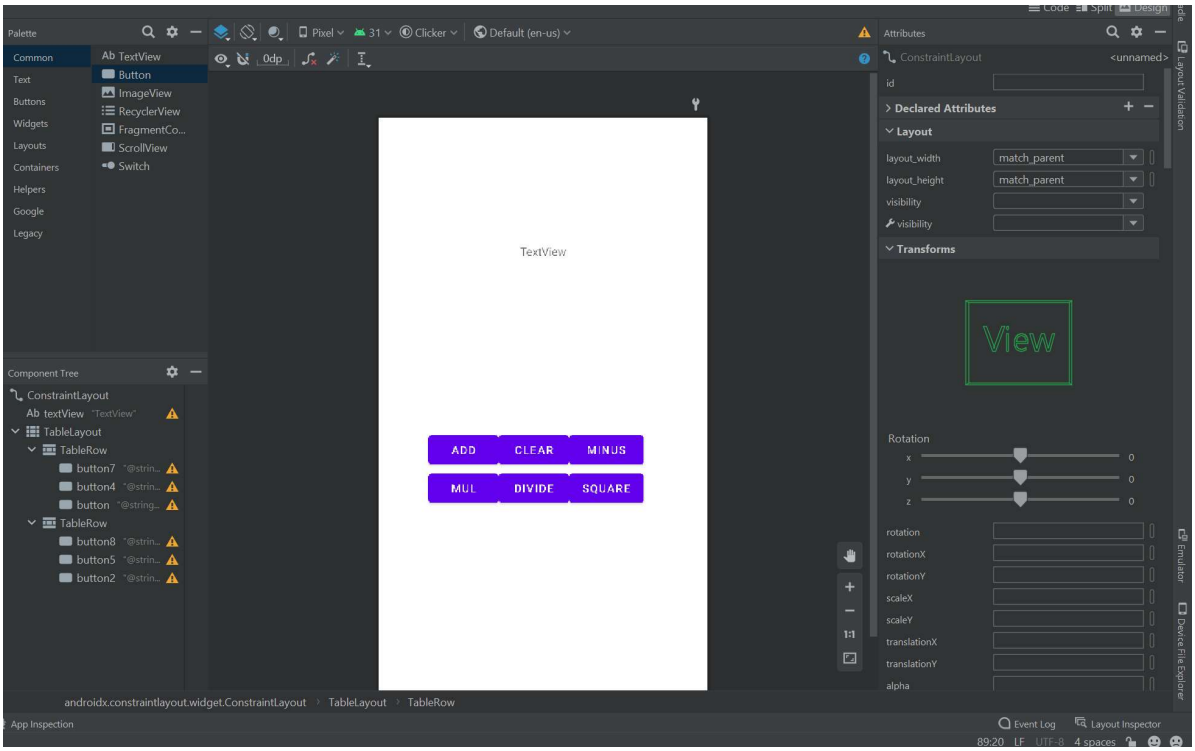
```
<RelativeLayout ... >
    <Button ... android:id="@+id/b1" android:text="B1"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />
    <Button ... android:id="@+id/b2" android:text="B2"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/b1" />
    <Button ... android:id="@+id/b3" android:text="B3"
        android:layout_centerHorizontal="true"
        android:layout_below="@+id/b2" />
    <Button ... android:id="@+id/b4" android:text="B4"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/b2" />
    <TextView ... android:id="@+id/tv1"
        android:text="I'm a TextView!"
        android:layout_centerInParent="true" />
    <Button ... android:id="@+id/b5" android:text="B5"
        android:padding="50dp"
        android:layout_centerHorizontal="true"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="50dp" />
</RelativeLayout>
```



Updating our Application with more buttons and layouts

Add more buttons with Table layout

We will now add 3 more buttons, and re-arrange them under a Table Layout. Your `activity_main.xml` should look something like this. For these new 3 buttons, their functions is **up to you**. The functions this guide will set **may be different from the one you choose to set**.



Similar as before, we will have to add the `onClick` method name and set `clickable` to True.

Making use of `strings.xml`

At this point, you would have realised it is quite tedious and messy to name your buttons in the `activity_main.xml` file. One solution is to make use of the file `strings.xml` to hold all our hard-coded values (like names of buttons). To make use of this, we have to change two things.

First, under `activity_main.xml`, `android:text="Some string"` to `android:text="@string/somename"`

Next, edit `strings.xml`. This is found under `res/Values/strings.xml`. To add a new string, simply add the line

```
<string name="somename">Clicker</string>
```

Your `activity_main.xml` should look similar to the code below. Note, we are only showing the first 3 buttons due to length restrictions. The remaining `TableRow` should be similar to the first `TableRow` with 3 buttons. Do ensure you add names and **onClick** methods.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
```

```
    <TextView
        android:id="@+id/textView"
        android:layout_width="235dp"
        android:layout_height="114dp"
        android:layout_marginStart="88dp"
        android:layout_marginTop="110dp"
        android:layout_marginEnd="88dp"
        android:gravity="center"
        android:text="TextView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TableLayout
        android:layout_width="286dp"
        android:layout_height="148dp"
        android:layout_marginStart="62dp"
        android:layout_marginEnd="63dp"
        android:layout_marginBottom="192dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent">

        <TableRow
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <Button
                android:id="@+id/button7"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:clickable="true"
                android:onClick="buttonAdd_click"
                android:text="@string/buttonAdd_click" />

            <Button
                android:id="@+id/button4"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:clickable="true"
                android:onClick="buttonClear_click"
                android:text="@string/buttonClear_click" />

            <Button
                android:id="@+id/button"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:clickable="true"
                android:onClick="buttonMinus_click"
                android:text="@string/buttonMinus_click" />
        </TableRow>

        <TableRow
            android:layout_width="match_parent"
            android:layout_height="match_parent">
            //truncated
        </TableRow>
```

```
    </TableLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Now, let us switch over to `strings.xml` for the naming of our buttons. Your xml file should look similar to the code below. Once again, the 3 new buttons added is **up to you** and our naming here is just a reference.

```
<resources>
    <string name="app_name">Clicker</string>
    <string name="buttonClear_click">Clear</string>
    <string name="buttonAdd_click">Add</string>
    <string name="buttonMinus_click">Minus</string>
    <string name="buttonDivide_click">Divide</string>
    <string name="buttonMult_click">Mul</string>
    <string name="buttonPow_click">Square</string>
</resources>
```

Programming button functions

Similar to before, we will have to declare new Methods for each button, and update the Count variable. Your code should look something similar to the following.

```
public class MainActivity extends AppCompatActivity {
    TextView text;
    int count = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        text = (TextView) findViewById(R.id.textView);
        text.setText(String.valueOf(count));
    }

    public void buttonClear_click(View view) {
        count = 0;
        text.setText(String.valueOf(count));
        Toast.makeText(this, "Count reset", Toast.LENGTH_LONG).show();
    }
    public void buttonMinus_click(View view) {
        count -= 1;
        text.setText(String.valueOf(count));
    }

    public void buttonAdd_click(View view) {
        count += 1;
        text.setText(String.valueOf(count));
    }

    public void buttonDivide_click(View view) {
        count /= 2;
        text.setText(String.valueOf(count));
    }
```

```
    public void buttonMult_click(View view) {
        count *= 2;
        text.setText(String.valueOf(count));
    }

    public void buttonPow_click(View view) {
        count = count * count;
        text.setText(String.valueOf(count));
    }
}
```

Be sure to save the code, and run the application to ensure it works!

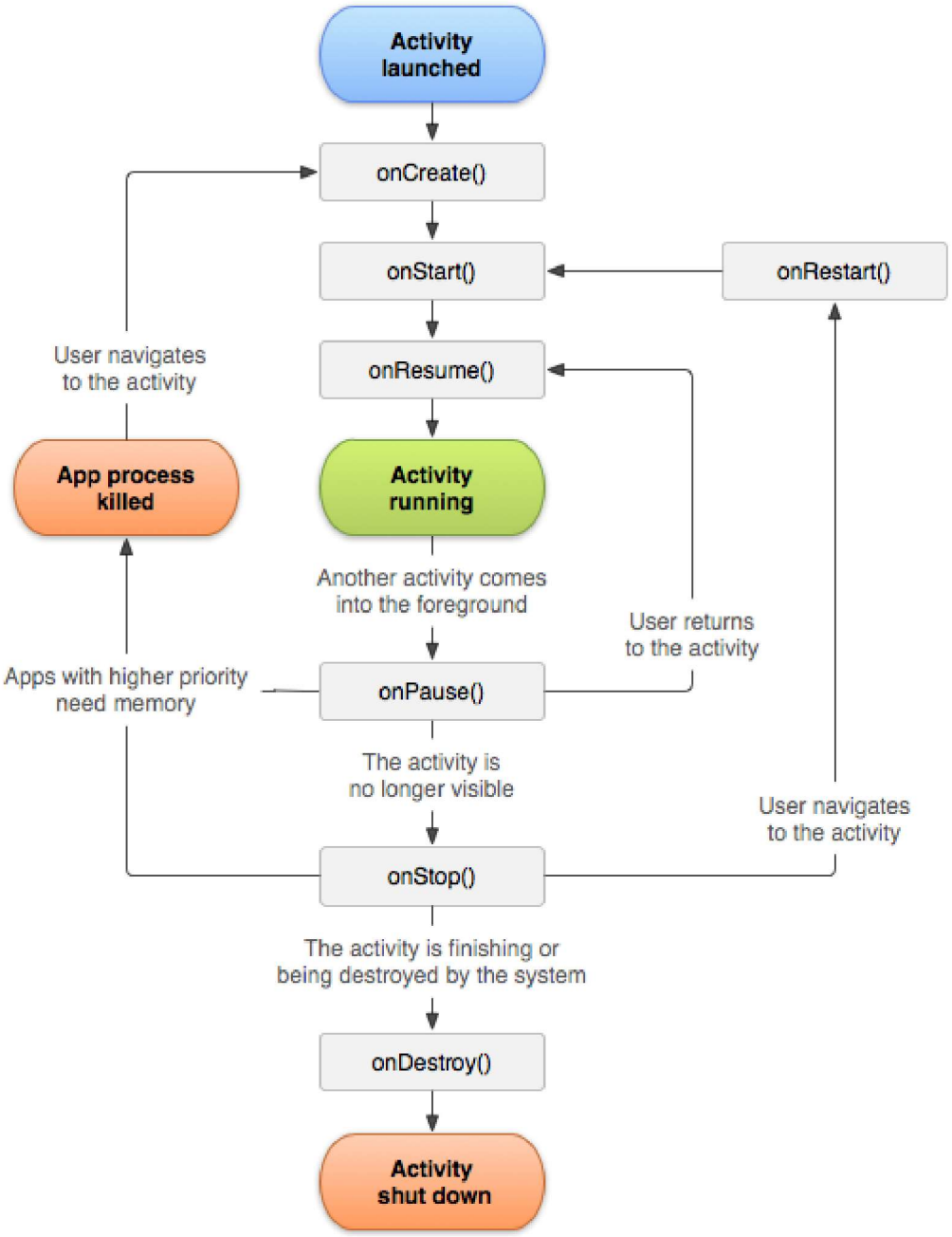
Activities

An Activity can be thought of as a "screen" or a "page". Before we learn how to use activities, let's go through some theory on the Activity lifecycle.

Activity Lifecycle

You might have noticed in the previous code examples that there is some code written in the `onCreate()` method. You might have also tried to rename the method/delete the method only to be met with errors. This is because of the activity lifecycle.

There are 7 states an Activity can be in. These 7 states make up the Activity Lifecycle from its creation to its destruction.

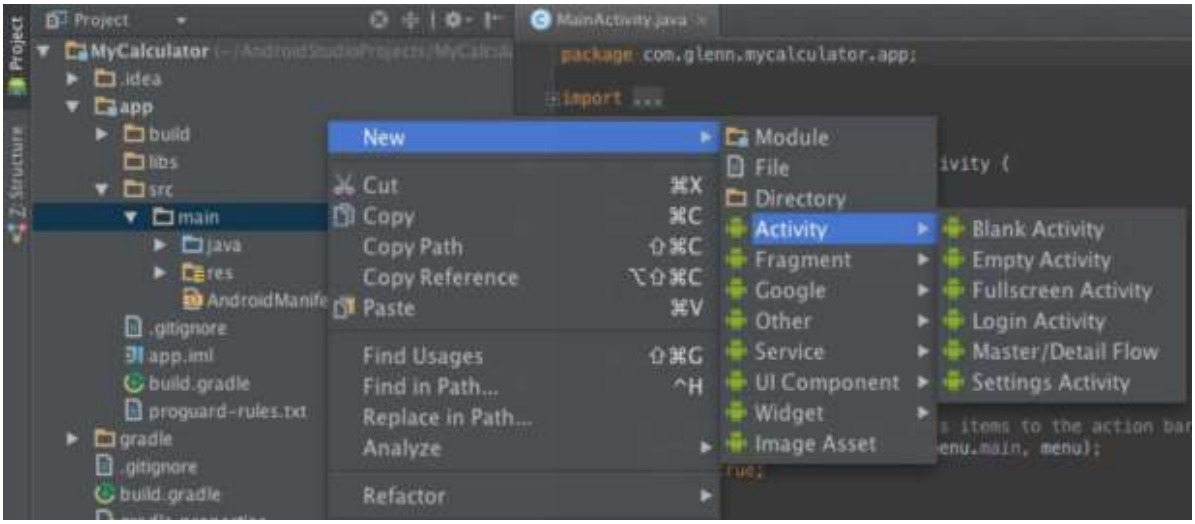


When the activity enters a new state, a method will be called (`onCreate()` is one such method). The 7 methods are:

Method	Description
`onCreate()`	Called when activity is first created. Used to set up the activity object, load resources such as images, layouts etc.
`onStart()`	Called when activity is becoming visible to the user. Usually used to reopen any resources you closed in `onStop()`
`onResume()`	Called when activity will start interacting with the user. May be used to initialize resources that were released in `onPause()`.
`onPause()`	Called when activity is not visible to the user. May be used to stop animations or release other system resources.
`onStop()`	Called when activity is no longer visible to the user. Performs heavy-duty shutdown tasks.
`onRestart()`	Called after your activity is stopped, prior to start. Useful if you want to run anything only after `onStop()` but not after `onCreate()`
`onDestroy()`	Called before the activity is destroyed

Adding a new Activity

To add an activity to the project, right-click "app" > New > Activity > Blank Activity.



This would create a `.java` class in `src/java` where we write our activity functions, and a `.xml` file in `res/layouts` where we define the graphical interface for our activity. An entry into `AndroidManifest.xml` is also entered automatically so that our app "knows" about the activity (this means if you rename your activity without updating the manifest, you might encounter an error).

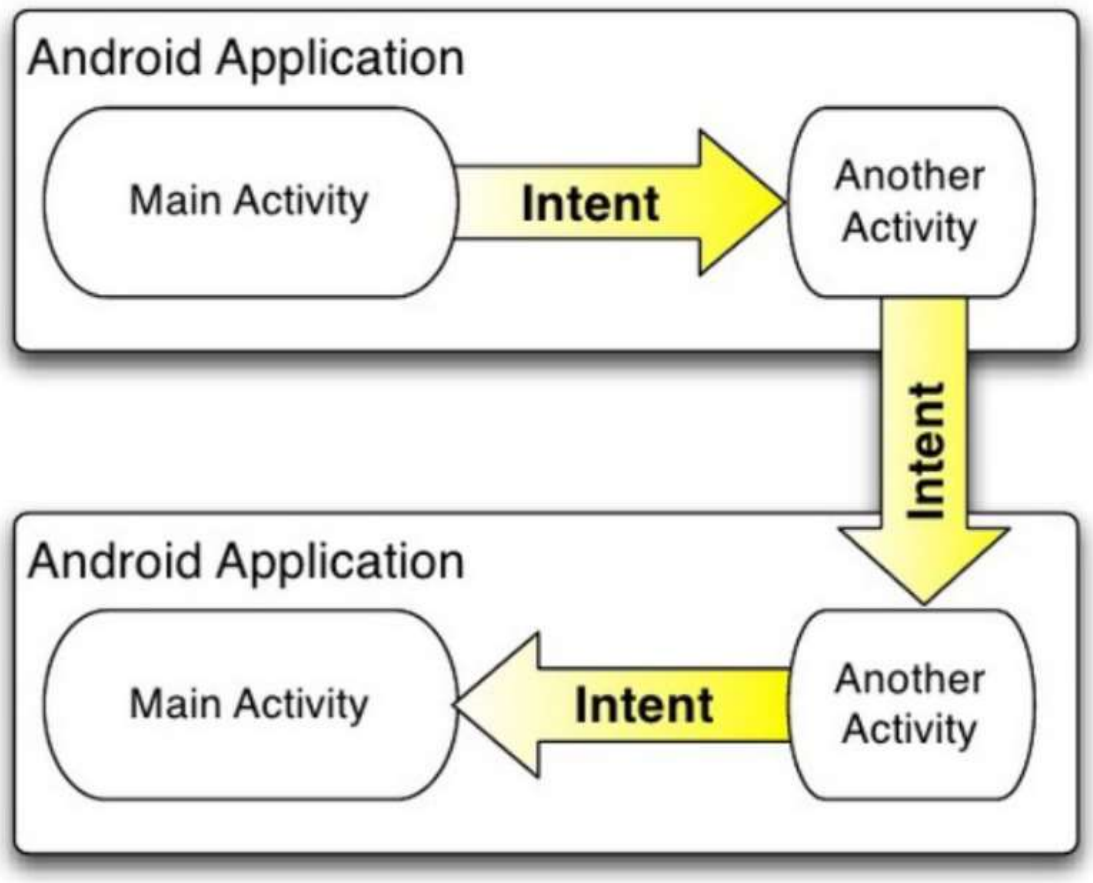
Intents

Background

Many applications have multiple activities. For example, in an address book app, the main activity can be a list of contacts, while another activity can be to view details of each contact. An activity A can launch another activity B in response to an event. Note, activities can pass data to each other. Activity A can pass data to Activity B, and Activity B can send data back to Activity A when it is done.

An intent is a bridge between activities. It serves as a way for one activity to invoke another.

- the activity can be in the same app or even in a different app
- can store extra data passed as "parameters" to the other activity
- new activity can "return" information back to the caller (original activity) if needed



Working with intents

To launch another activity (usually in response to an event such as a button click), create an Intent object and call `startActivity` with it.

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

To pass additional parameters or data to the second activity, `putExtra` is called. Extra data is stores as key-value pairs, similar to a Map.

```
Intent intent = new Intent(this, ActivityName.class);
intent.putExtra("Key", value);
startActivity(intent);
```

In the new activity that was started, you can grab extra data passed to it. `getIntent` accesses the Intent that spawned the activity. The intent has methods such as `getExtra`, `getIntExtra`, `getStringExtra` to extract additional data stored inside the intent.

For example,

```
public class SecondActivity extends Activity {
    ...
    public void onCreate(Bundle savedState) {
        super.onCreate(savedState);
        setContentView(R.layout.activity_second)
        //Get Intent and get extra data passed
        Intent intent = getIntent();
        String extra = intent.getExtra("name");
    }
}
```

If the calling activity wants to wait for a result from the called activity, `startActivityForResult` should be used instead of using `startActivity`.
`startActivityForResult`:

- requires a unique ID number to represent the action performed
- requires a final int constant with a value of your choice

To send back a result from a called activity,

1. an intent has to be created to go back.
2. Extra data should be passed using `putExtra`.
3. `setResult` must be called
4. finish by calling `onDestroy`

For example,

```
public class SecondActivity extends Activity {
    ...
    public void returnClick(View view){
        Intent intent = new Intent()
        intent.putExtra("Key", value);
        setResult(RESULT_OK, intent);
        finish(); //calls onDestroy
    }
}
```

An example is included below which creates an Intent, and gets back the result from the called activity.

```
public class FirstActivity extends Activity {
    private static final int REQ_CODE = 123; //Must be from 0 to 65535

    public void buttonClick(View view) {
        Intent intent = getIntent(this, SecondActivity.class);
        startActivityForResult(intent, REQ_CODE);
    }

    protected void onActivityResult(int requestCode, int resultCode, Intent
intent) {
        super.onActivityResult(requestCode, resultCode, intent);
        if (requestCode == REQ_CODE) {
            //came back from SecondActivity
            String data = intent.getStringExtra("Key")
        }
    }
```

```
}
```

Adding Win screen to our application

Updating Our Application

For now, let's update our `main.java` to keep track of the number of times the user has pressed the button. When the count hits 100, we will send the user to the Win screen as well as display the number of times the User has pressed a button before he hit 100.

To do this, we will

1. Initialise a new variable named `press`
2. Increment `press` every time a button is pressed.

Your `main.java` should look something like this

```
public class MainActivity extends AppCompatActivity {
    TextView text;
    int count = 0;
    int press = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        text = (TextView) findViewById(R.id.textView);
        text.setText(String.valueOf(count));
    }

    public void buttonClear_click(View view) {
        count = 0;
        press++;
        text.setText(String.valueOf(count));
        Toast.makeText(this, "Count reset", Toast.LENGTH_LONG).show();
    }
    public void buttonMinus_click(View view) {
        count -= 1;
        press++;
        text.setText(String.valueOf(count));
    }

    public void buttonAdd_click(View view) {
        count += 1;
        press++;
        text.setText(String.valueOf(count));
    }

    public void buttonDivide_click(View view) {
        count /= 2;
        press++;
        text.setText(String.valueOf(count));
    }

    public void buttonMult_click(View view) {
        count *= 2;
        press++;
```

```
        text.setText(String.valueOf(count));
    }

    public void buttonPow_click(View view) {
        count = count * count;
        press++;
        text.setText(String.valueOf(count));
    }
}
```

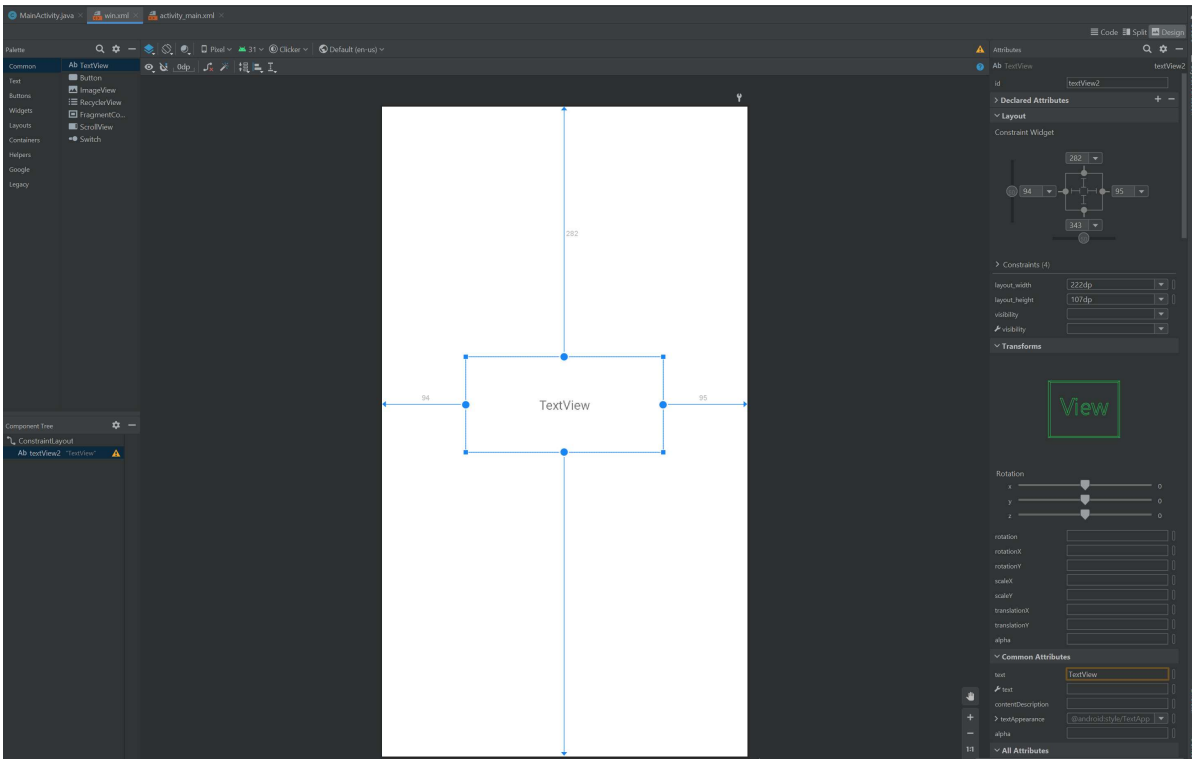
Now we can move on to adding our new Win Activity and Win Screen.

Addition of new Win Activity UI

First, let's create our new Win Screen UI.

Under `res/layout`, we will create a new `xml` file called `win.xml`. This time, we will only be adding a TextView in the center of the screen. Do remember that TextView should have **gravity** set to `center`.

Your `win.xml` should look something like this.



Your XML code should look something like this

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="222dp"
        android:layout_height="107dp"
```

```
        android:layout_marginStart="94dp"
        android:layout_marginTop="282dp"
        android:layout_marginEnd="95dp"
        android:layout_marginBottom="343dp"
        android:text="TextView"
        android:gravity="center"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Addition of new Win Activity Class

Now, we will create a new Class. We will name this `WinActivity.java` and it should be created in the same location as `MainActivity.java`. Our initial code should contain the following

```
package com.example.clicker;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;

public class WinActivity extends AppCompatActivity {
    TextView text;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.win);
        text = (TextView) findViewById(R.id.textView);
    }
}
```

Pay close attention to `onCreate()` and ensure that the `ContentView` has been set correctly to our new layout `win.xml`. Similar to our code in `main.java`, we interact with our TextView by calling `findViewById` to access it's view object.

For this new Activity to be recognised by our Android Application, we have to edit the `AndroidManifest.xml` file and add the following line

```
<activity android:name=".WinActivity" ></activity>
```

Your `AndroidManifest.xml` should look something like this.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.clicker">

    <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/Theme.Clicker">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

In this way, you should be able to call `WinActivity.java` from our `MainActivity.java`

Linking both via Intents

Now, let us code the Intent Call. Intents will allow us to start a new Activity (`WinActivity`) as well as pass data to it, such as the number of times a User has pressed a button.

Let us first edit `MainActivity.java`. Our goal is to

1. Check if score has hit 100
2. If so, create a new Intent and pass the number of times a button has been pressed to it
3. Create a new Activity with this intent.

Remember, we can start a new instance of an Activity by passing an Intent to `startActivity()`.

For this,

- We will first import `import android.content.Intent;`
- We will create a function to check the current count. If `count == 100`, we will start our win Activity. The function should look something like this. Note, `i.putExtra` allows us to pass additional data to the new Activity in the form of a **Key-Value pair**. This means that our value press is linked to the key "press".

```
private void checkCount() {
        if (count == 100){
            Intent i = new Intent(MainActivity.this, WinActivity.class);
            i.putExtra("press",press);
            startActivity(i);
        }
    }
```

- We will call this function `checkCount()` every time a button is pressed. Our final code for `MainActivity.java` should look something like this. Note, the code shown below has been **truncated to show only the first 3 buttons**.

```
public class MainActivity extends AppCompatActivity {
    TextView text;
    int count = 0;
    int press = 0;
```

```
    private void checkCount() {
        if (count == 100){
            Intent i = new Intent(MainActivity.this, WinActivity.class);
            i.putExtra("press",press);
            startActivity(i);
        }
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        text = (TextView) findViewById(R.id.textView);
        text.setText(String.valueOf(count));
    }

    public void buttonClear_click(View view) {
        count = 0;
        press++;
        text.setText(String.valueOf(count));
        Toast.makeText(this, "Count reset", Toast.LENGTH_LONG).show();
    }
    public void buttonMinus_click(View view) {
        count -= 1;
        press++;
        text.setText(String.valueOf(count));
        checkCount();
    }

    public void buttonAdd_click(View view) {
        count += 1;
        press++;
        text.setText(String.valueOf(count));
        checkCount();
    }

}
```

Now, let us edit `WinActivity.java`. Our goal is to

1. Retrieve the value of `press`
2. Display a Text to tell the user he has won with X number of presses

For this, we will:

- Add a new String variable named `message`
- Update `onCreate` with the following code

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.win);
        Bundle extras = getIntent().getExtras();
        if (extras != null) {
            int press = extras.getInt("press");
            message = "You won! You took " + press + " number of button press to
win!";
```

```
            //The key argument here must match that used in MainActivity.java
        }
        text = (TextView) findViewById(R.id.textView);
        text.setText(message);

    }
```

Note, `Bundle extras = getIntent().getExtras()` allows us to retrieve our value passes as an Extra Data in `MainActivity.java`.

Your final `WinActivity.java` should look something like this:

```
package com.example.clicker;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;

public class WinActivity extends AppCompatActivity {
    TextView text;
    String message;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.win);
        Bundle extras = getIntent().getExtras();
        if (extras != null) {
            int press = extras.getInt("press");
            message = "You won! You took " + press + " number of button press to
win!";
            //The key argument here must match that used in MainActivity.java
        }
        text = (TextView) findViewById(R.id.textView);
        text.setText(message);

    }
}
```

The End

Github Link: https://github.com/dbsqwerty/android-app-dev-tutorial

Congratulations, you have reached the end of this set of notes. You can refer to the link above in case you missed out something, or would like to refer to the final code. We hope that this set of notes would help in your coursework, IA and your future projects. Some places to refer to for additional help include Stack Overflow (https://stackoverflow.com/), the official Android Documentation (https://developer.android.com/docs) and Google.

Credits

- Sean Seah
- River Koh

References

- https://developer.android.com/reference
- https://web.stanford.edu/class/cs193a/