Android Deep Dive With Dr. Chuck Easttom

Android Programming

- Originally Apps were written in Java, but they also allow C/C++ extensions. The Android Development Studio also supports the Koltin programming language.
- Since the release of Android Studio 3.0 in October 2017, Kotlin is included as an alternative to the standard Java compiler. The Android Kotlin compiler lets the user choose between targeting Java 6 or Java 8 compatible bytecode.

Kotlin

- Kotlin works with the Java Virtual Machine, and is sponsored by JetBrains and Google. There is a Kotlin Foundation. Since Android Studio 3.0 (October 2017), Kotlin has been an alternative for Java. Google began using Kotlin for Android Apps in May 2019. The Kotlin language was first released in 2011.
- Here is the Kotlin Hello World

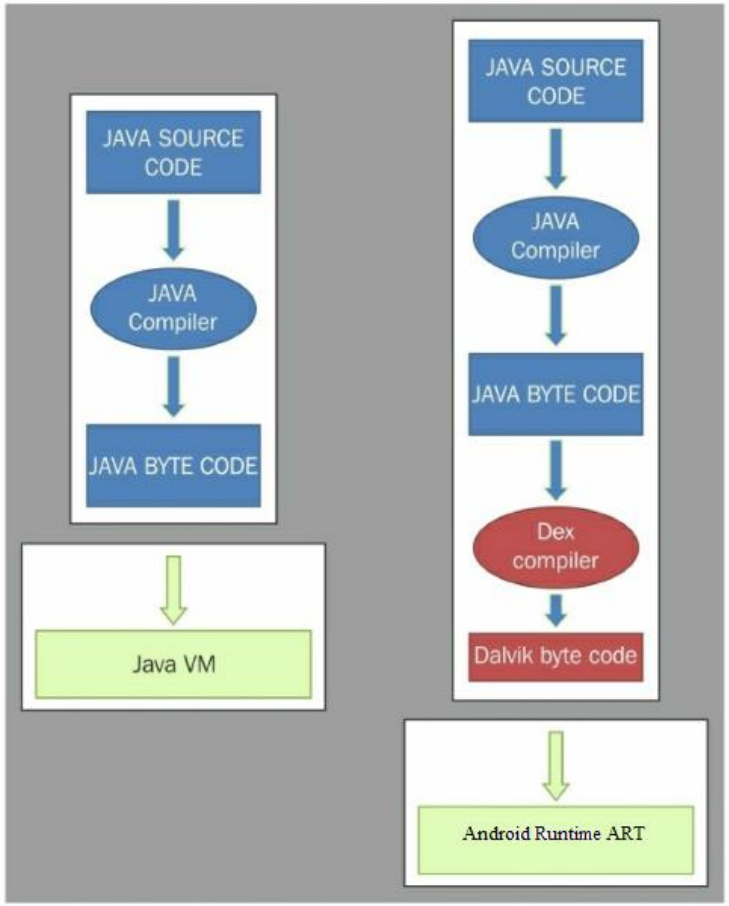
```
/**
 * We declare a package-level function main which returns Unit and takes
 * an Array of strings as a parameter. Note that semicolons are optional.
 */

fun main(args: Array<String>) {
    println("Hello, world!")
}
```

Kotlin

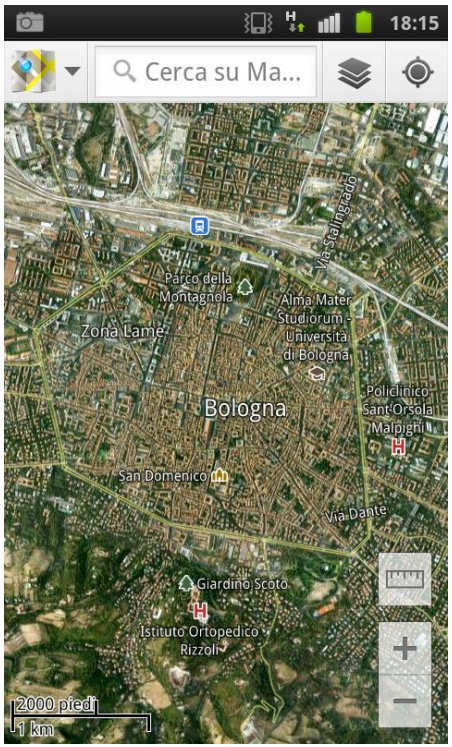
- Kotlin does have powerful OOP features. You can add methods to a class,w ithout first creating a derived class. It is called an extension method
- package MyStringExtensions
- fun String.lastChar(): Char = get(length - 1)
- >>> println("Kotlin".lastChar())
- Free online Kotlin course at https://play.kotlinlang.org/byExample/overview

Compiling

- When a Java program is compiled, we get byte code
- A Java virtual machine (JVM) (a virtual machine is an application that acts as an operating system) can execute this byte code
- In Android, this Java byte code is further converted to Dalvik byte code by the dex compiler
- This Dalvik byte code is then fed into Dalvik virtual machine (DVM) which can read and use the code

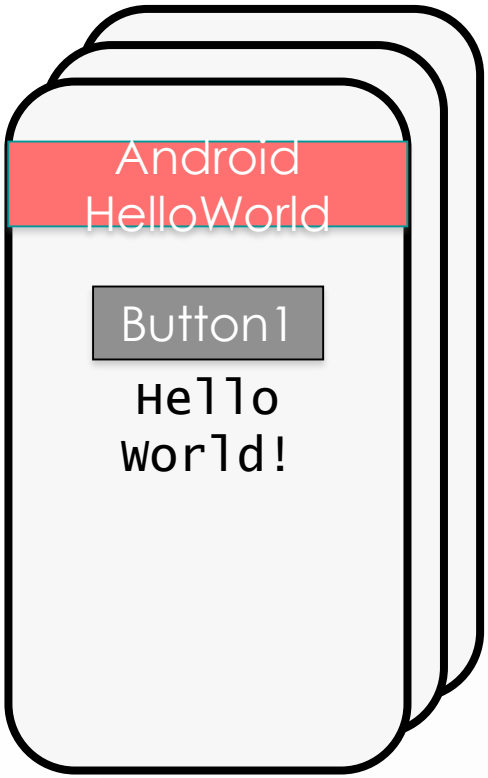


Android Applications



- **Activities**
- **Intents**
- **Services**
- **Content Providers**
- **Broadcast Receivers**

Android Components:



- An Activity corresponds to a single screen of the Application.
- An Application can be composed of *multiples screens* (Activities).
- The **Home Activity** is shown when the user launches an application.
- Different activities can exhange information one with each other.

Android Components:

- Each activity is composed by a list of *graphics components*.
- Some of these components (also called **Views**) can interact with the user by handling **events** (e.g. Buttons).
- Two ways to build the graphic interface:

PROGRAMMATIC APPROACH

Example:

```
Button button=new Button (this);
TextView text= new TextView();
text.setText("Hello world");
```


Android Components:

- Each activity is composed by a list of *graphics components*.
- Some of these components (also called **Views**) can interact with the user by handling **events** (e.g. Buttons).
- Two ways to build the graphic interface:

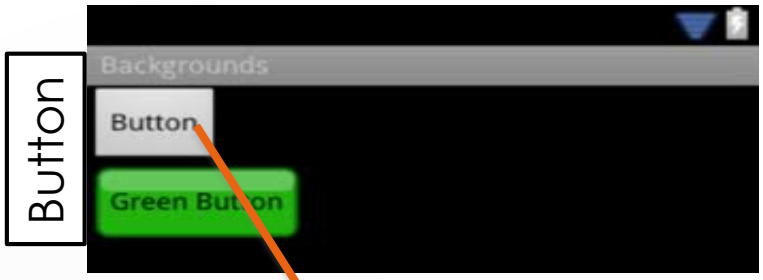
DECLARATIVE APPROACH

Example:

```
< TextView android.text=@string/hello" android:textcolor=@color/blue
android:layout_width="fill_parent" android:layout_height="wrap_content" />
< Button android.id="@+id/Button01" android:textcolor="@color/blue"
android:layout_width="fill_parent" android:layout_height="wrap_content" />
```

Android Components:

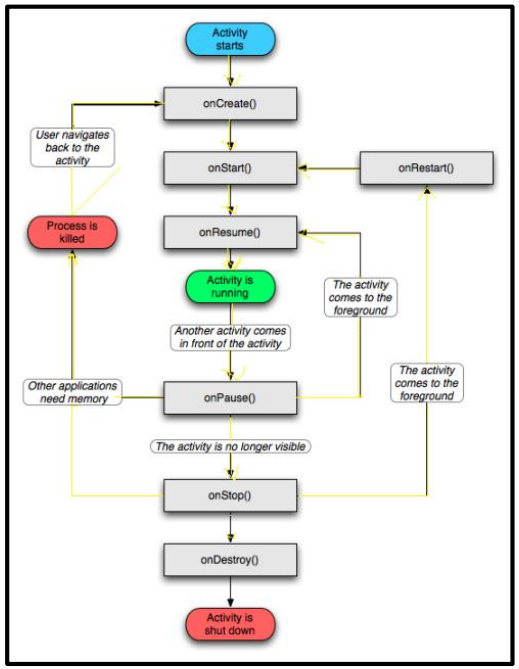
- **Views** can generate **events** (caused by human interactions) that must be managed by the Android-developer.



ESEMPIO

```
public void onClick(View arg0) {
     if (arg0 == Button) {
          // Manage Button events
     }
}
```

Android Components:



- The **Activity Manager** is responsible for creating, destroying, managing activities.
- Activities can be on different **states**: *starting*, *running*, *stopped*, *destroyed*, *paused*.
- Only one activity can be on the **running** state at a time.
- Activities are organized on a **stack**, and have an event-driven life cycle

Android Components:

```
public class MyApp extends Activity {


    public void onCreate() { ... }
    public void onPause()  { ... }
    public void onStop()   { ... }
    public void onDestroy(){ ... }
    ….
}
```

Called when the Activity is **created** the first time.

Called when the Activity is **partially visible**.

Called when the Activity is **no longer visible**.

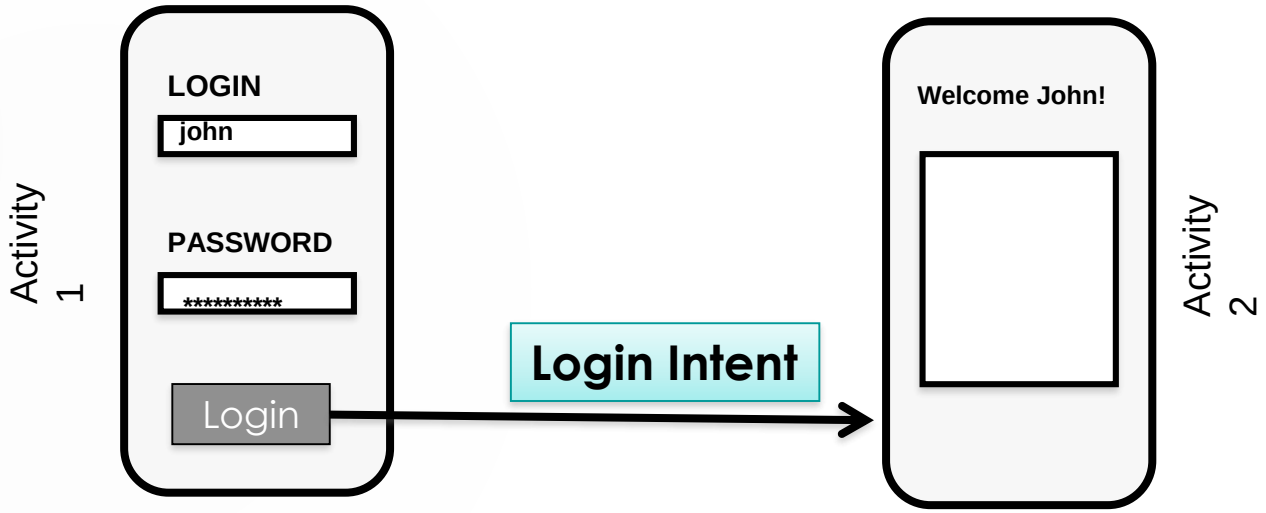
Called when the Activity is **dismissed**.

Intent

- Intent is a goal action component which takes care of the process of inter-components communication (ICC)
- Intent is simply a message object containing a destination component address and data
- Protection
- Each application executes as its own user identity, such that OS provides system-level isolation;
- Android middleware contains a reference monitor that mediates the inter-component communication (ICC).

Android Components:

- **Intents**: asynchronous **messages** to activate core Android components (e.g. Activities).
- **Explicit** Intent → The component *(e.g. Activity1)* specifies the destination of the intent *(e.g. Activity 2)*.

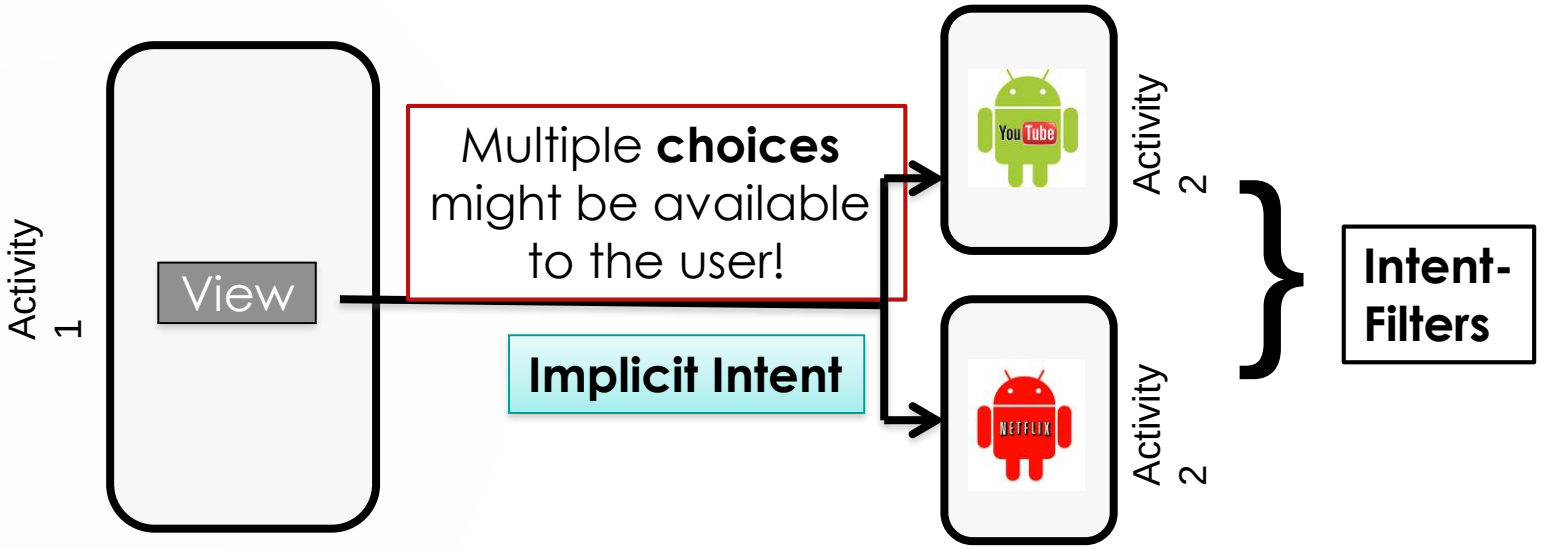


Intent Filters

- Used to determine recipient of Intent
- Specify the main entrance for activities
- A user interface consists of a series of Activities
- Each Activity is a “screen”.
- Intent may leave a security flaw (hole)
  - Solution: Intents explicitly define receiver

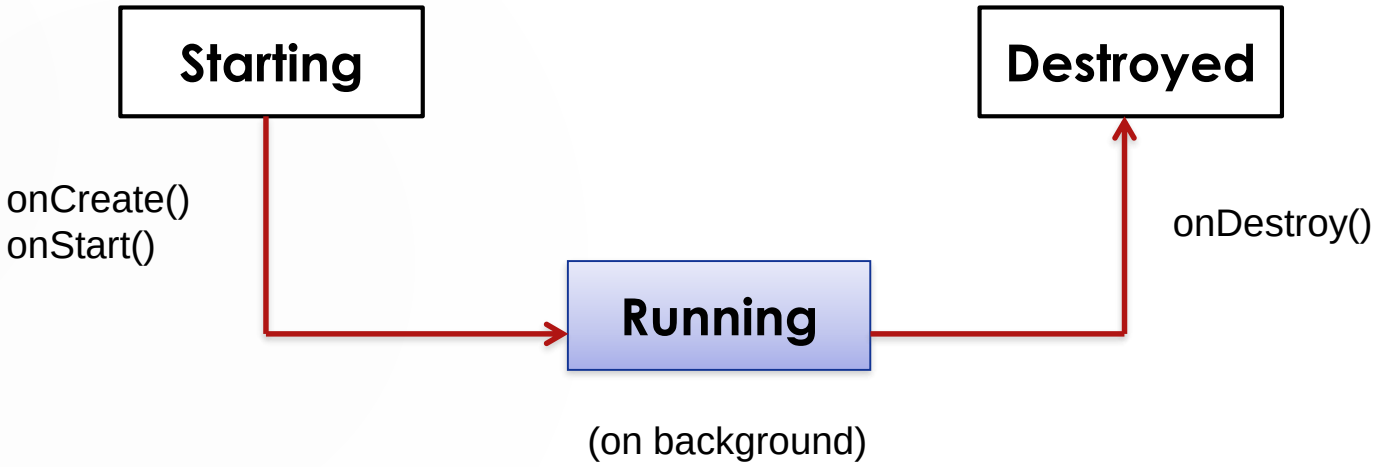
Android Components:

- **Intents**: asynchronous **messages** to activate core Android components (e.g. Activities).
- **Implicit** Intent → The component *(e.g. Activity1)* specifies the type of the intent *(e.g. "View a video")*.



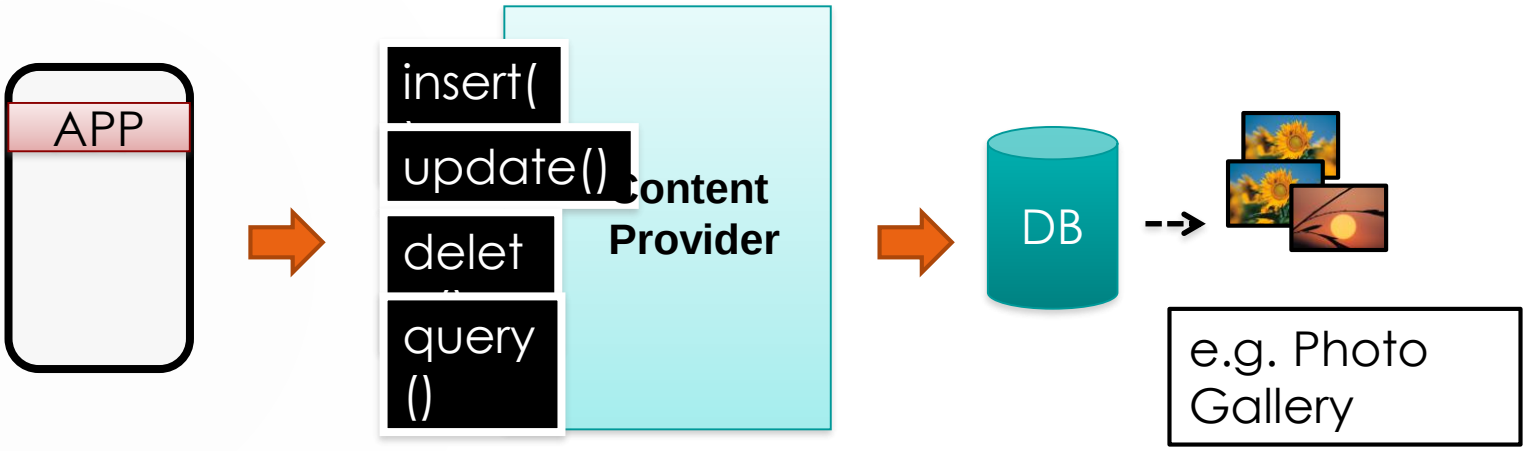
Android Components:

- **Services**: Run in **background** and do not provide an user interface.
- Used for **non-interactive** tasks (e.g. networking).
- Service life-time composed of 3 states:



Android Components:

- Each Android **application** has its own **private** set of data (managed through *files* or through *SQLite* database).
- **Content Providers**: Standard **interface** to *access and share data among different applications*.



Content Providers

- Makes some of the application data available to other applications
- It's the only way to transfer data between applications in Android (no shared files, shared memory, pipes, etc.)
- Extends the class ContentProvider;
- Other applications use a ContentResolver object to access the data provided via a ContentProvider

Broadcast Receivers

- Receive and react to broadcast announcements
- Extend the class BroadcastReceiver
- Examples of broadcasts:
  - Low battery, power connected, shutdown, timezone changed, etc.
  - Other applications can initiate broadcasts

Android Components:

BROADCAST RECEIVER example

```
class WifiReceiver extends BroadcastReceiver {
        public void onReceive(Context c, Intent intent) {
            String s = new StringBuilder();
            wifiList = mainWifi.getScanResults();
            for(int i = 0; i < wifiList.size(); i++){
                s.append(new Integer(i+1).toString() + ".");
                s.append((wifiList.get(i)).toString());
                s.append("\\n");
            }
            mainText.setText(sb);
        }
    }
```

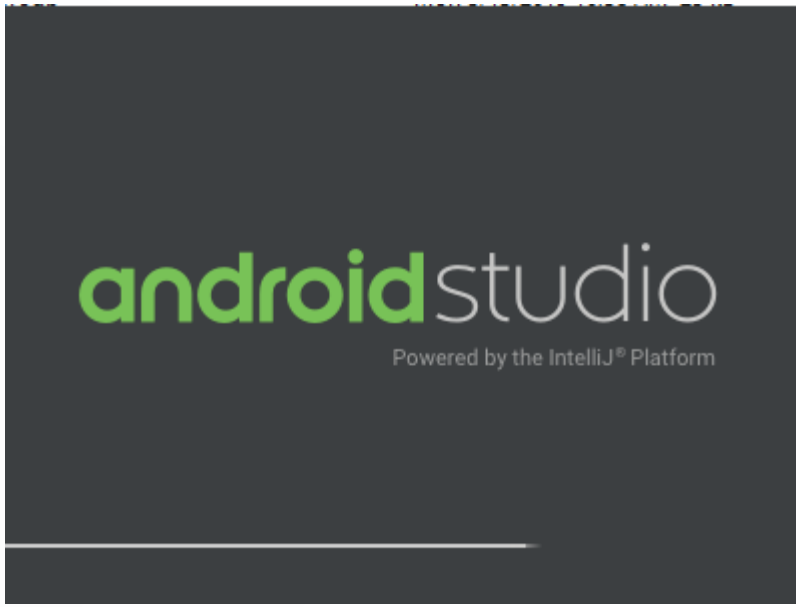

Android Manifest

- Its main purpose in life is to declare the components to the system:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity
android:name="com.example.project.FreneticActivity"
                android:icon="@drawable/small_pic.png"
                android:label="@string/freneticLabel"
                . . . >
        </activity>
        . . .
    </application>
</manifest>
```

Android Studio

- https://developer.android.com/studio



Example Manifest with Activity

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity
android:name="com.example.project.FreneticActivity
"
                  android:icon="@drawable/small_pi
c.png"
                  android:label="@string/freneticL
abel"
                  . . .  >
        </activity>
        . . .
    </application>
</manifest>
```

- The name attribute of the <activity> element names the Activity subclass that implements the activity.
- The icon and label attributes point to resource files containing an icon and label that can be displayed to users to represent the activity.

Example Manifest with Activity

```
<?xml version="1.0" encoding="utf-8"?>
 <manifest . . . >
     <application . . . >
         <activity
 android:name="com.example.project.FreneticActivity"
                   android:icon="@drawable/small_pic.png"
                   android:label="@string/freneticLabel"
                   . . .  >
         </activity>
         . . .
     </application>
 </manifest>
```

- The other components are declared in a similar way — <service> elements for services, <receiver> elements for broadcast receivers, and <provider> elements for content providers.
- Activities, services, and content providers that are not declared in the manifest are not visible to the system and are consequently never run.
- Broadcast receivers can either be declared in the manifest, or they can be created dynamically in code (as BroadcastReceiver objects) and registered with the system by calling Context.registerReceiver().

Intent Filters

- An Intent object can explicitly name a target component. If it does, Android finds that component (based on the declarations in the manifest file) and activates it.
- If a target is not explicitly named, Android must locate the best component to respond to the intent.
- Android does so by comparing the Intent object to the *intent filters* of potential targets.
- A component's intent filters inform Android of the kinds of intents the component is able to handle. Like other essential information about the component, they're declared in the manifest file.

Intent Filters in a Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticActivity"
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . .  >
            <intent-filter . . . >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
            <intent-filter . . . >
                <action android:name="com.example.project.BOUNCE" />
                <data android:mimeType="image/jpeg" />
                <category android:name="android.intent.category.DEFAULT"
/>
            </intent-filter>
        </activity>
        . . .
    </application>
</manifest>
```

- The first filter — the combination of the action "android.intent.action.MAIN" and the category "android.intent.category.LAUNCHER" — marks the activity as one that should be represented in the application launcher, the screen listing applications users can launch on the device.
- The activity is the entry point for the application, the initial one users would see when they choose the application in the launcher.

Intent Filters in a Manifest

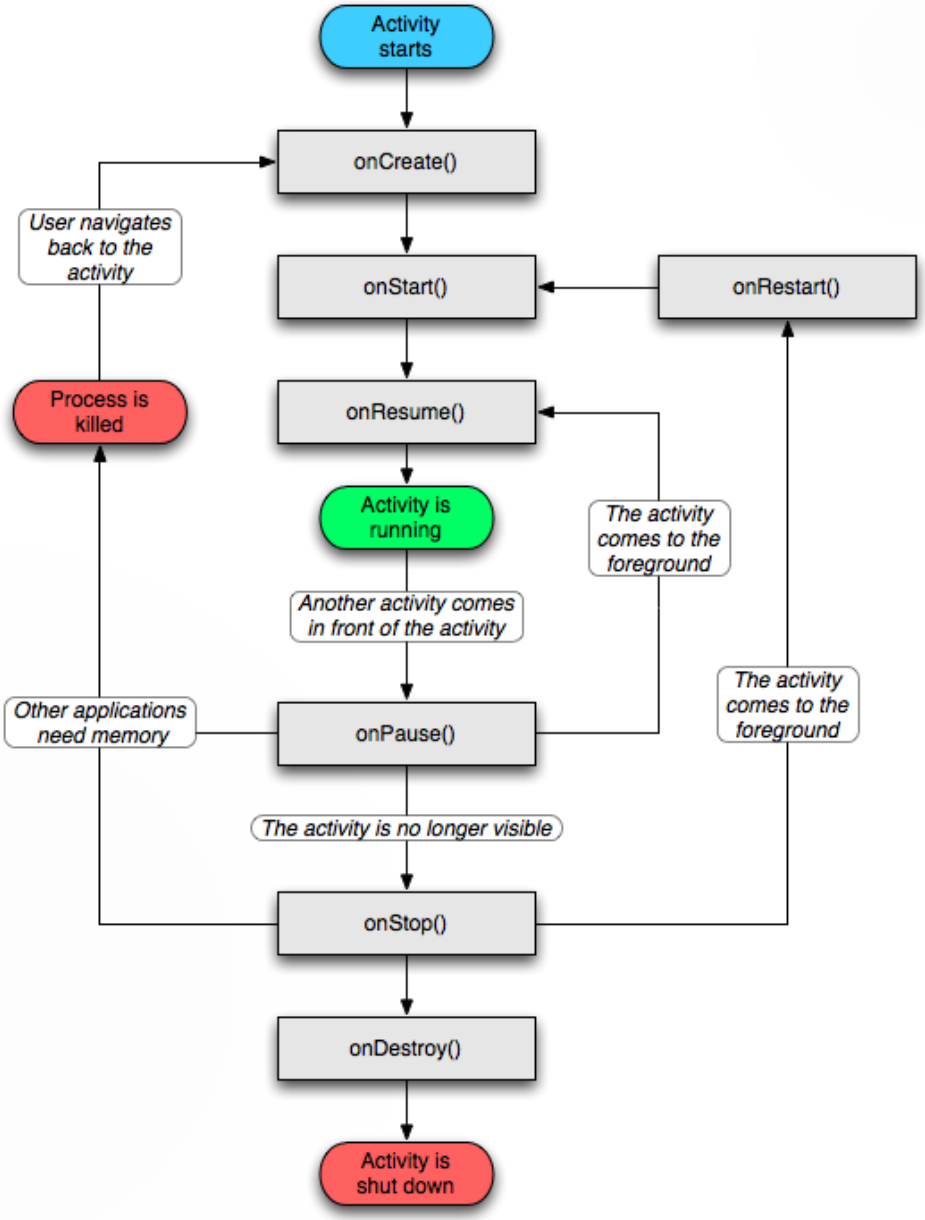
```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticActivity"
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . .  >
            <intent-filter . . . >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter . . . >
                <action android:name="com.example.project.BOUNCE" />
                <data android:mimeType="image/jpeg" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
        . . .
    </application>
</manifest>
```

- The second filter declares an action that the activity can perform on a particular type of data.
- A component can have any number of intent filters, each one declaring a different set of capabilities. If it doesn't have any filters, it can be activated only by intents that explicitly name the component as the target.
- For a broadcast receiver that's created and registered in code, the intent filter is instantiated directly as an IntentFilter object. All other filters are set up in the manifest.

Intent Filters in a Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticActivity"
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . .  >
            <intent-filter . . . >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter . . . >
                <action android:name="com.example.project.BOUNCE" />
                <data android:mimeType="image/jpeg" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
        . . .
    </application>
</manifest>
```

- The second filter declares an action that the activity can perform on a particular type of data.
- A component can have any number of intent filters, each one declaring a different set of capabilities. If it doesn't have any filters, it can be activated only by intents that explicitly name the component as the target.
- For a broadcast receiver that's created and registered in code, the intent filter is instantiated directly as an IntentFilter object. All other filters are set up in the manifest.


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

onCreate()

- Called when the activity is first created.
- This is where you should do all of your normal static set up — create views, bind data to lists, and so on.
- Passed a Bundle object containing the activity's previous state, if that state was captured.
- Always followed by onStart().
- Not killable.

onRestart()

- Called after the activity has been stopped, just prior to it being started again.
- Always followed by onStart().
- Not killable.

onResume()

- Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack, with user input going to it.
- Always followed by onPause().
- Not killable.

onPause()

- Called when the system is about to start resuming another activity.
- This method is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on.
- It should do whatever it does very quickly, because the next activity will not be resumed until it returns.
- Followed either by onResume() if the activity returns back to the front, or by onStop() if it becomes invisible to the user.
- Killable.

onStop()

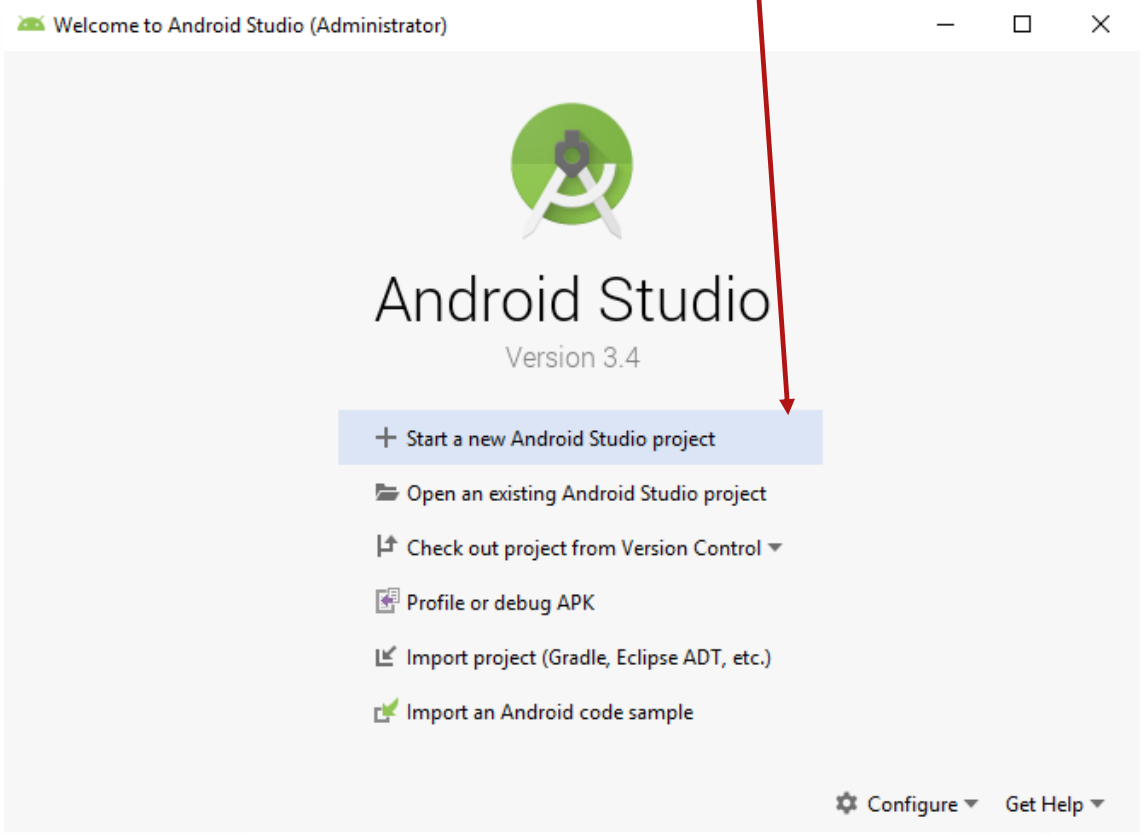
- Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it.
- Followed either by onRestart() if the activity is coming back to interact with the user, or by onDestroy() if this activity is going away.
- Killable.

onDestroy()

- Called before the activity is destroyed. This is the final call that the activity will receive.
- It could be called either because the activity is finishing (someone called finish() on it), or because the system is temporarily destroying this instance of the activity to save space.
- You can distinguish between these two scenarios with the isFinishing() method.
- Killable.

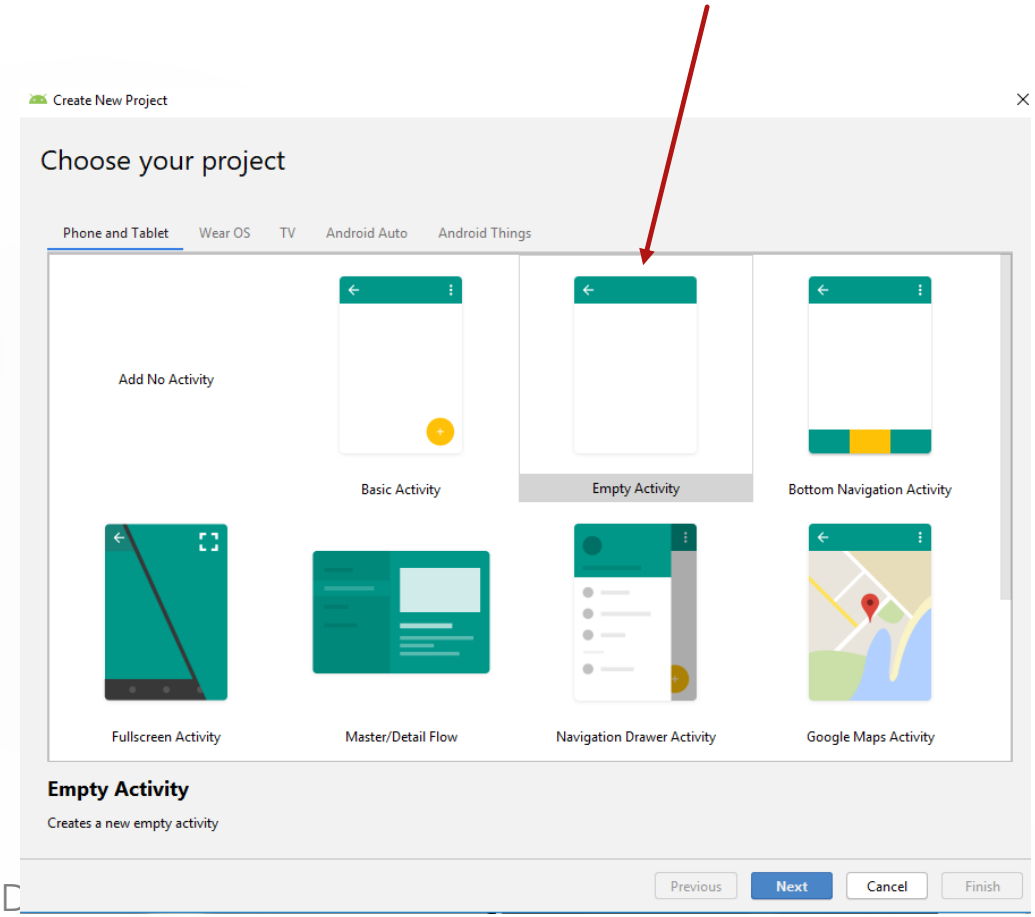
First Screen

- We will be creating a new project



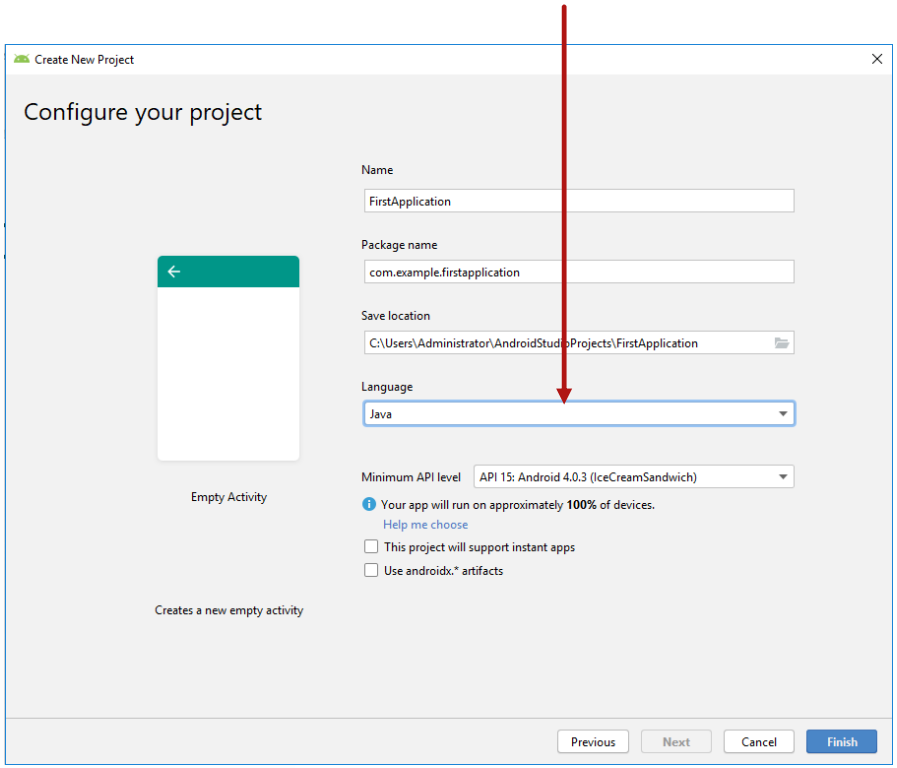
Your first project

We will start with an empty project

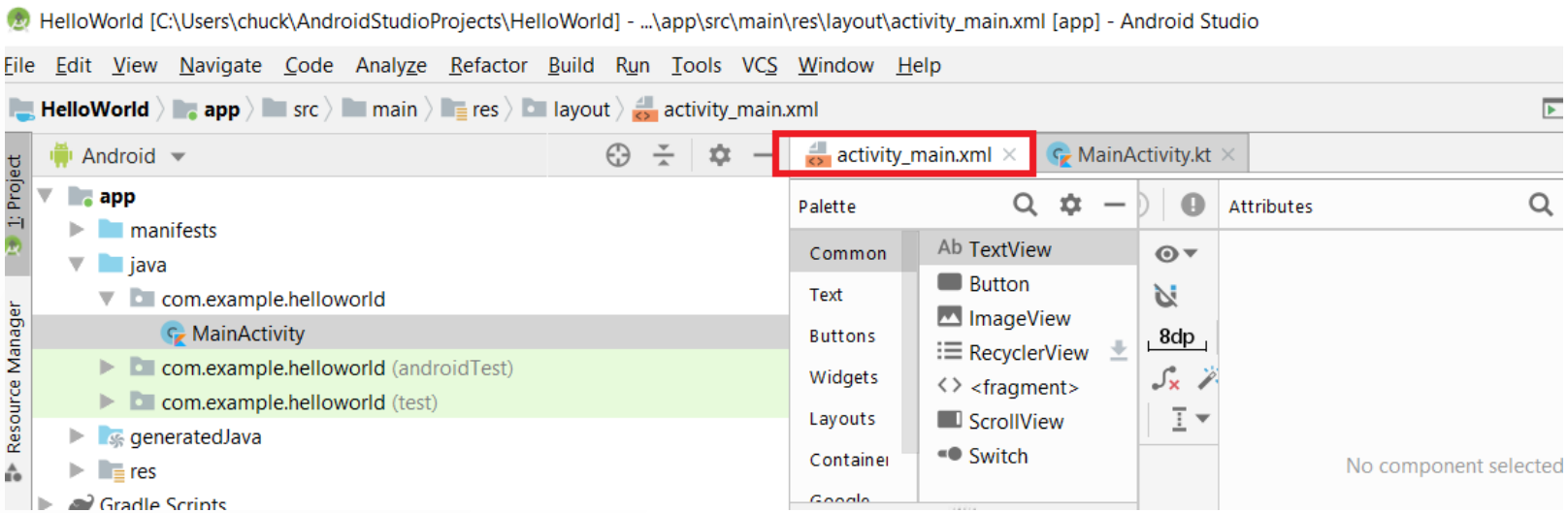


Your first project

Change the name and select Java for the language

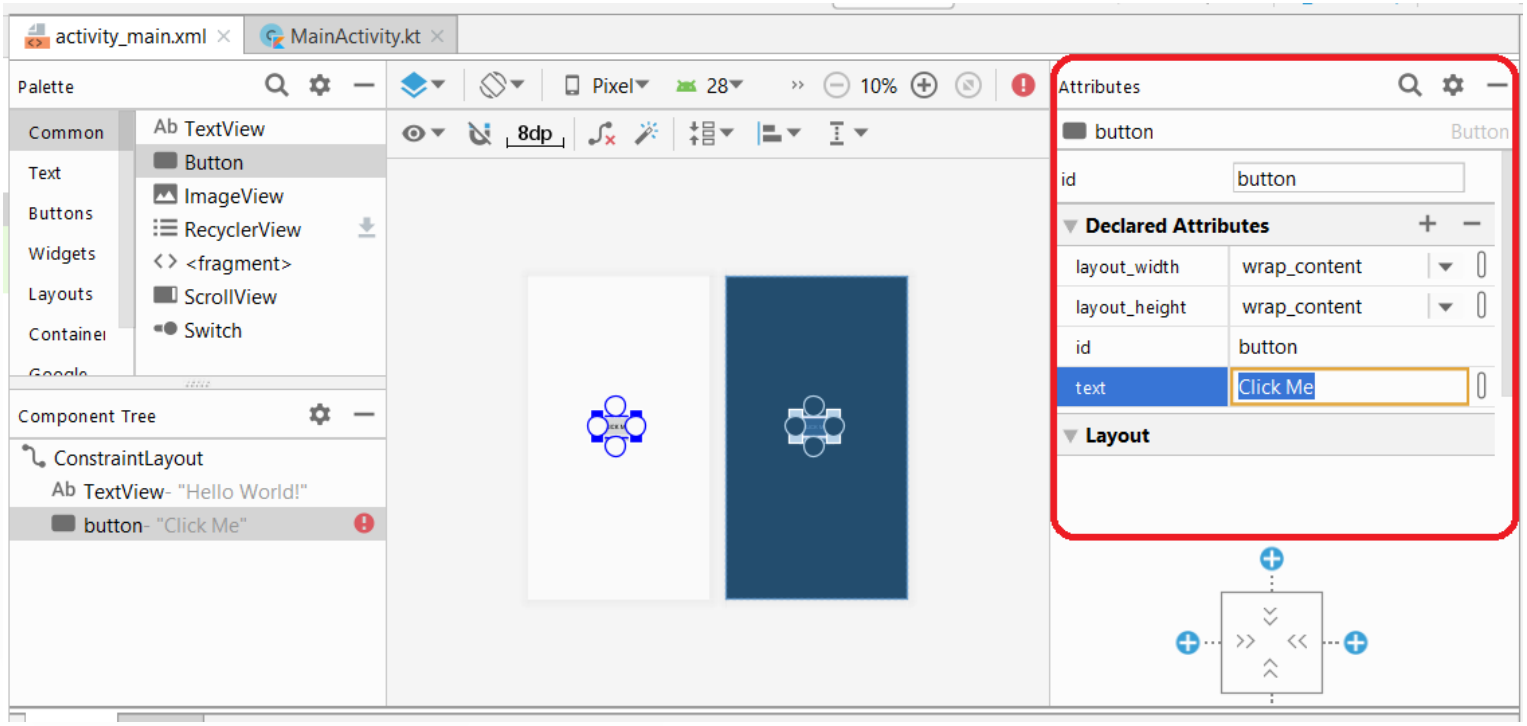


Navigating in the IDE

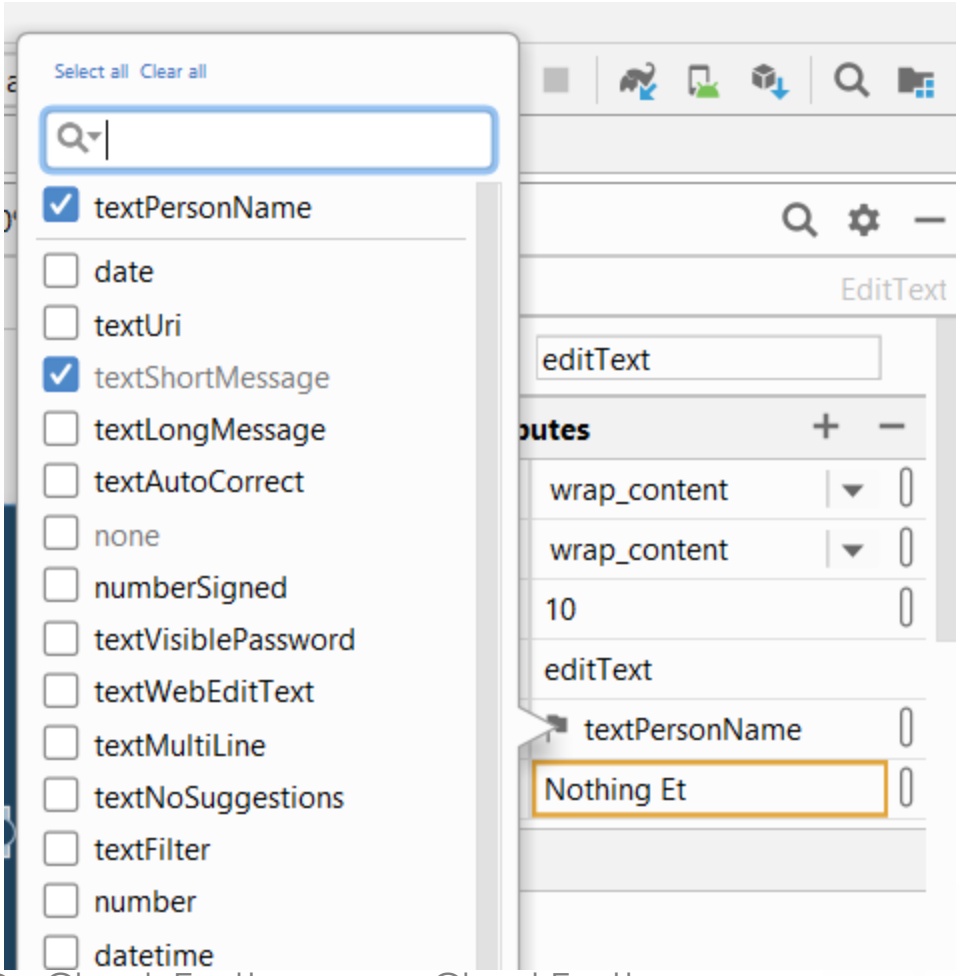


Android

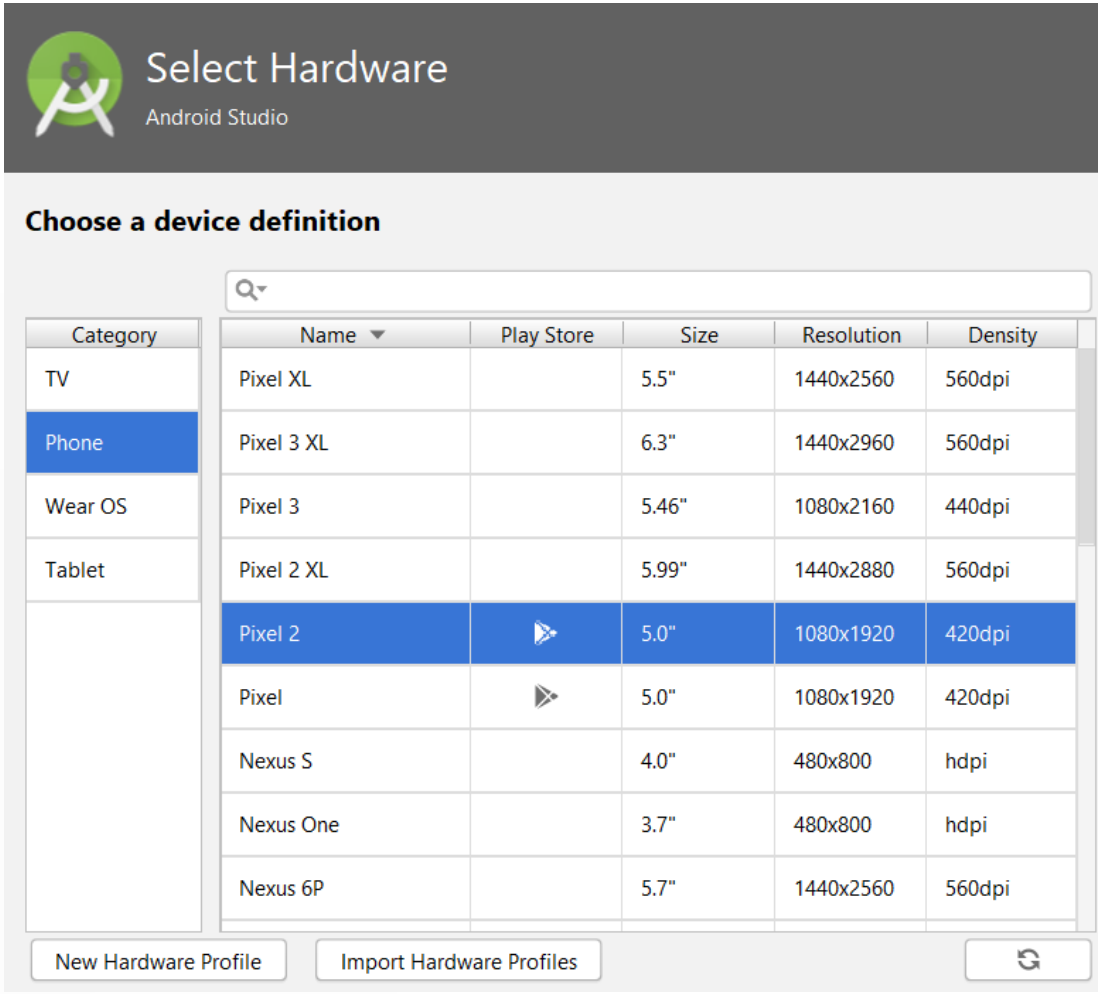
Finding properties



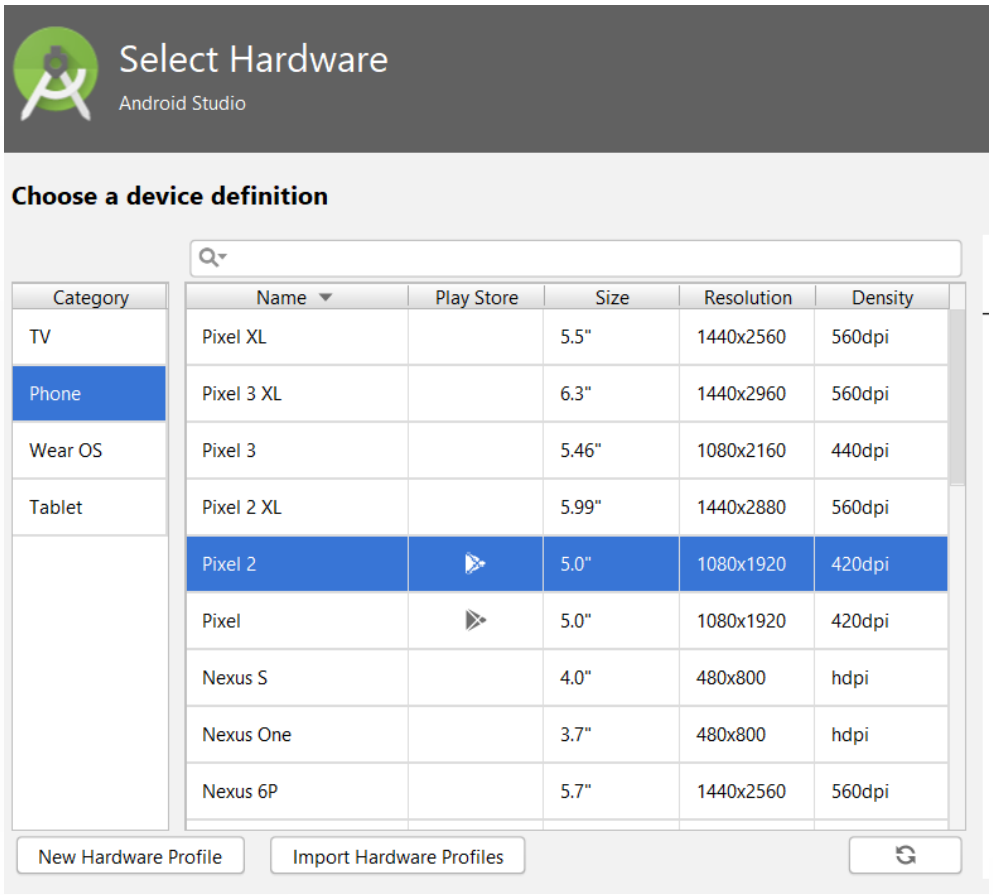
Look a little closer



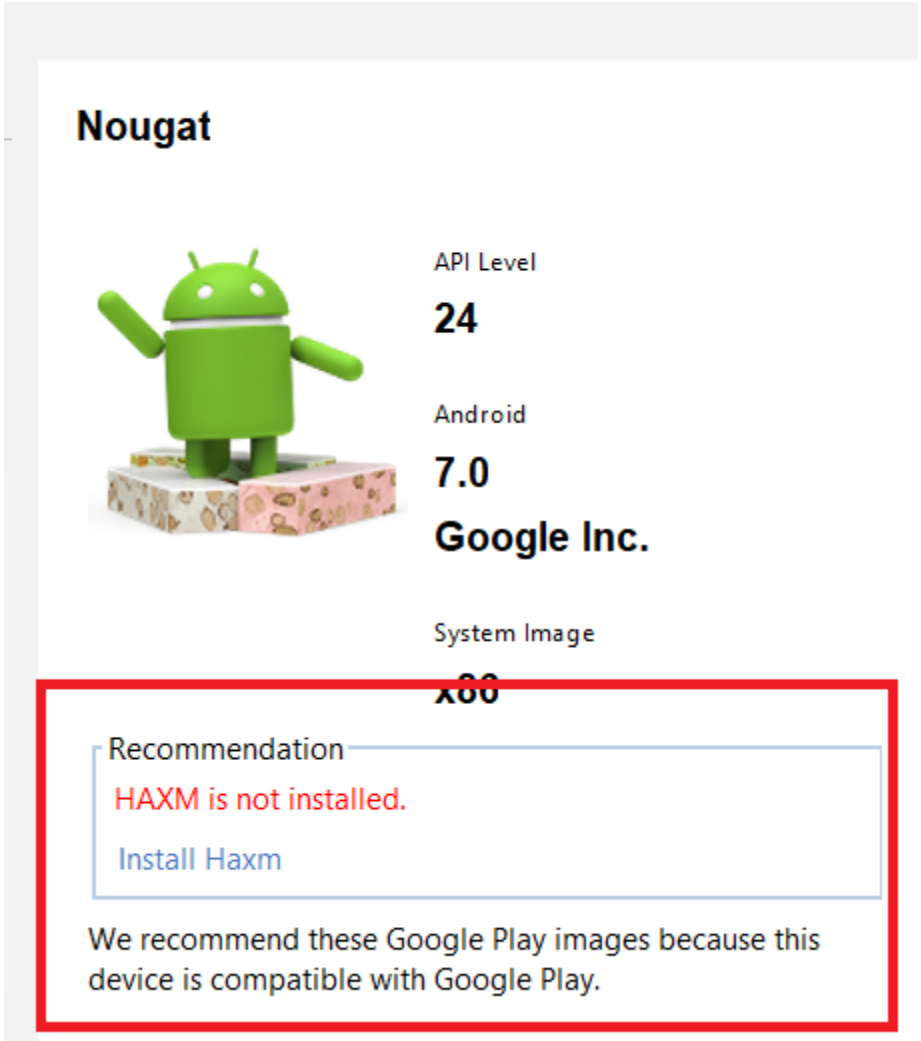
Virtual Devices



You get to choose the virtual device

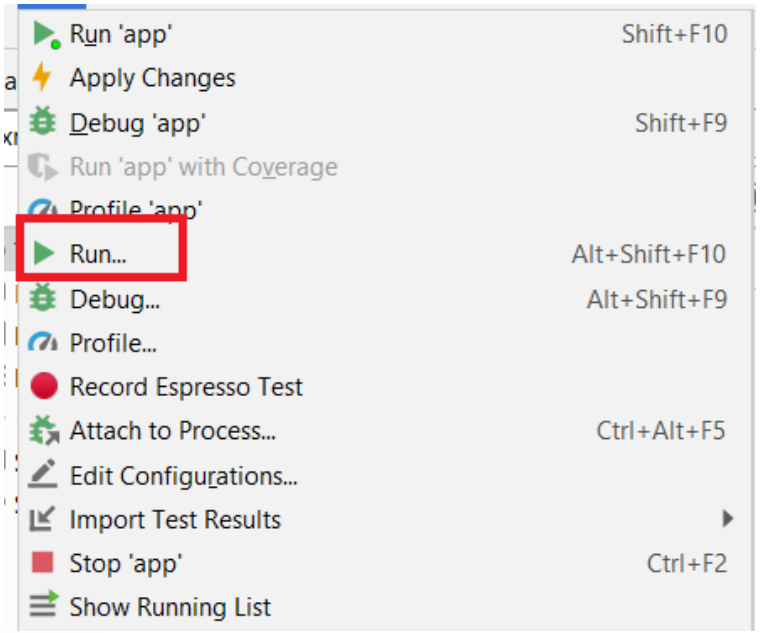


If you see this install it!

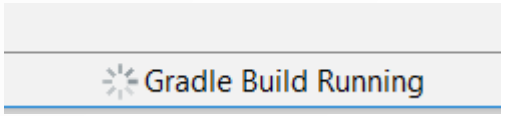


Selecting Run

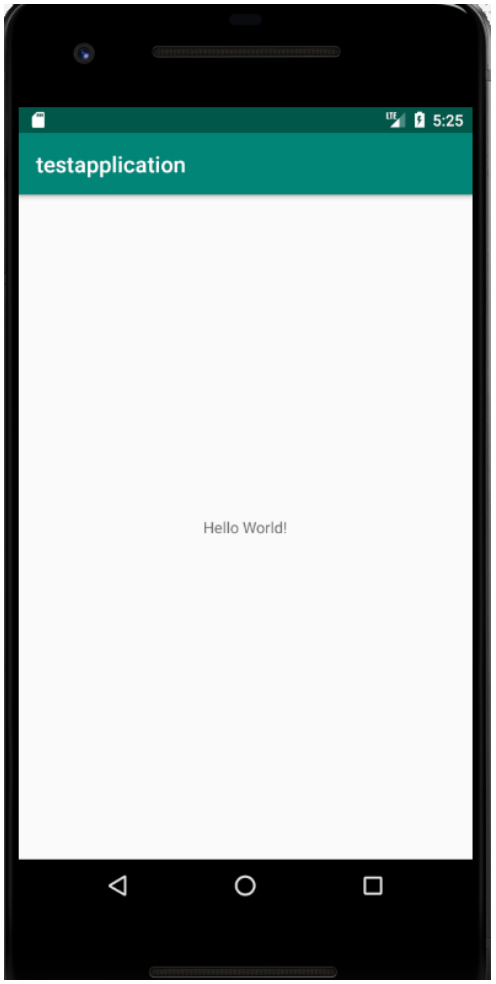
- Just press this



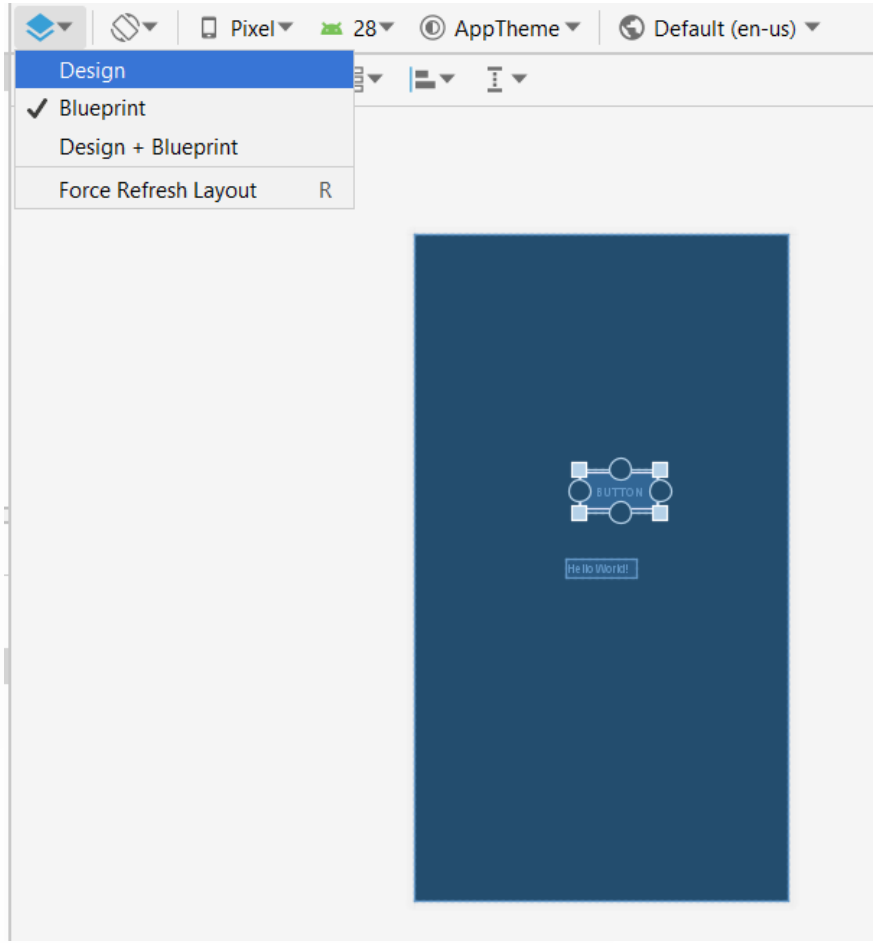
- And you will see this

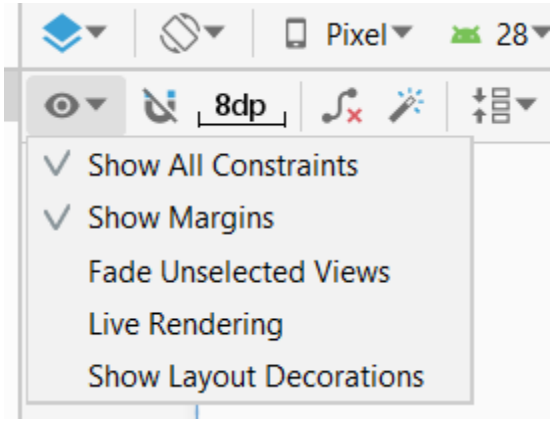


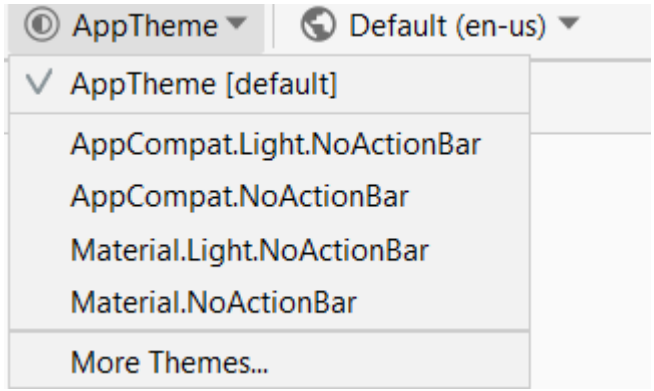
Then there is your app



Choosing Layout





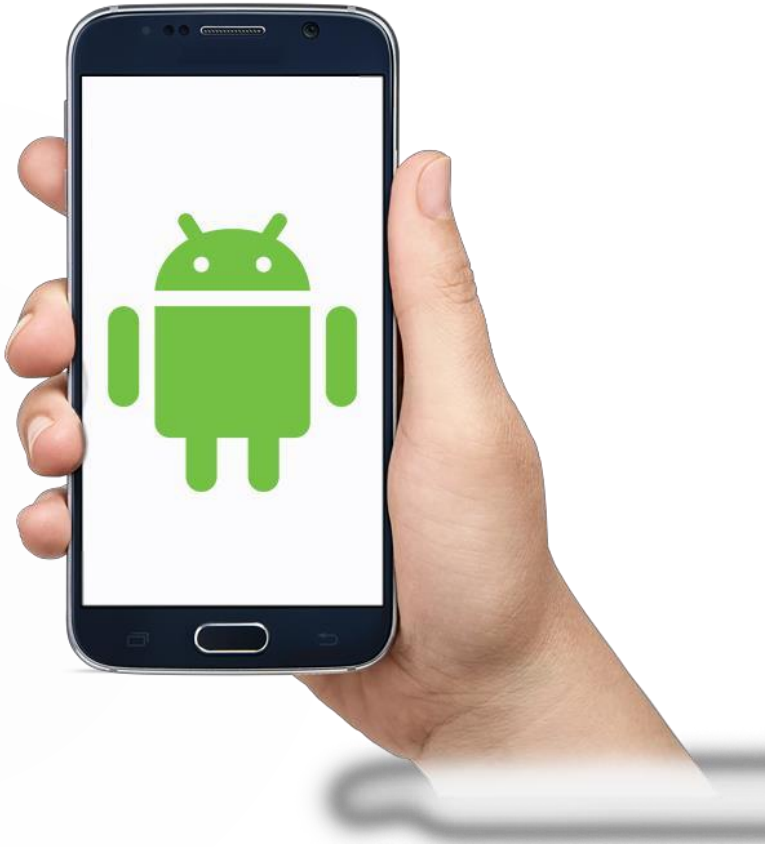


Make the code happen

Note:Android Studio is helpful. You will get the chance to click 'alt + enter' to get suggested code solutions

```
package com.example.helloworld;

import ...

public class MainActivity extends AppCompatActivity {

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

        //Code the button
        // 2 steps: get button
        Button  btnOne = findViewById(R.id.button);
        final TextView txtHello = findViewById(R.id.txtHello);
        btnOne.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                txtHello.setText("Howdy");
            }
        });
        // //then do stuff when it is clicked
    }
}
```


Basic Android Apps



App Manifest

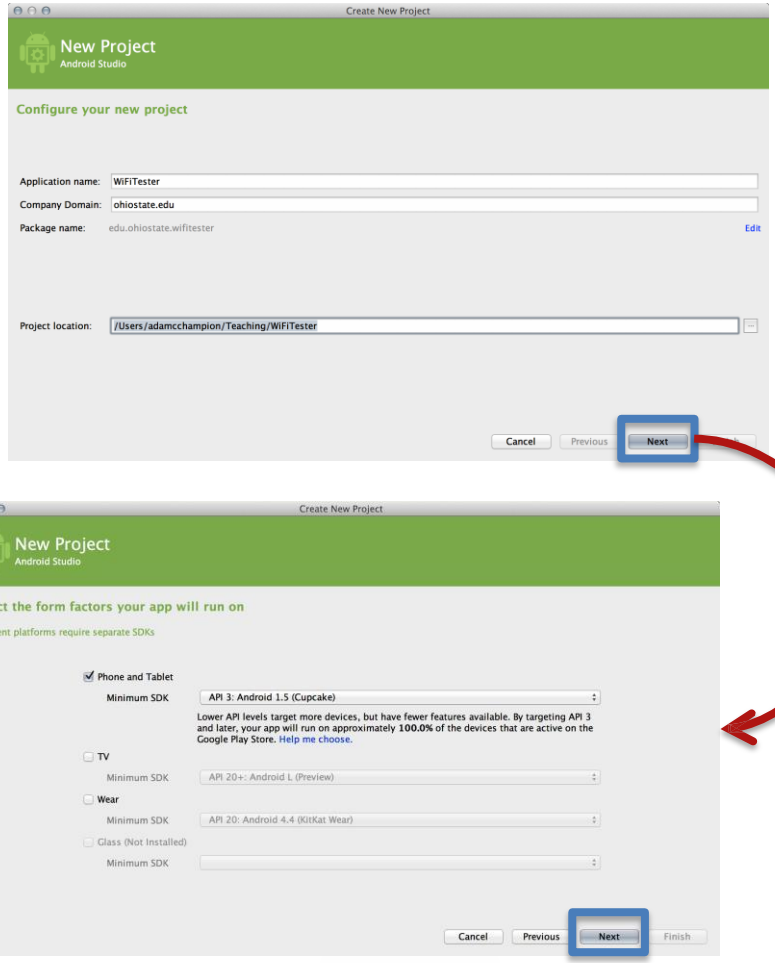
- Every Android app must include an AndroidManifest.xml file describing functionality
- The manifest specifies:
  - App's Activities, Services, etc.
  - Permissions requested by app
  - Minimum API required
  - Hardware features required, e.g., camera with autofocus
  - External libraries to which app is linked, e.g., Google Maps library

App Creation Checklist

- If you own an Android device:
- Ensure drivers are installed
- Enable developer options on device under *Settings*, specifically *USB Debugging*
- Android 4.2+: Go to *Settings→About phone*, press *Build number* 7 times to enable developer options
- For Android Studio:
- Under File→*Settings→Appearance,* enable "Show tool window bars"; the *Android* view shows LogCat, devices
- Programs should log states via android.util.Log's Log.d(APP_TAG_STR, "debug"), where APP_TAG_STR is a final String tag denoting your app
- Other commands: Log.e() (error); Log.i() (info); Log.w() (warning); Log.v() (verbose) – same parameters

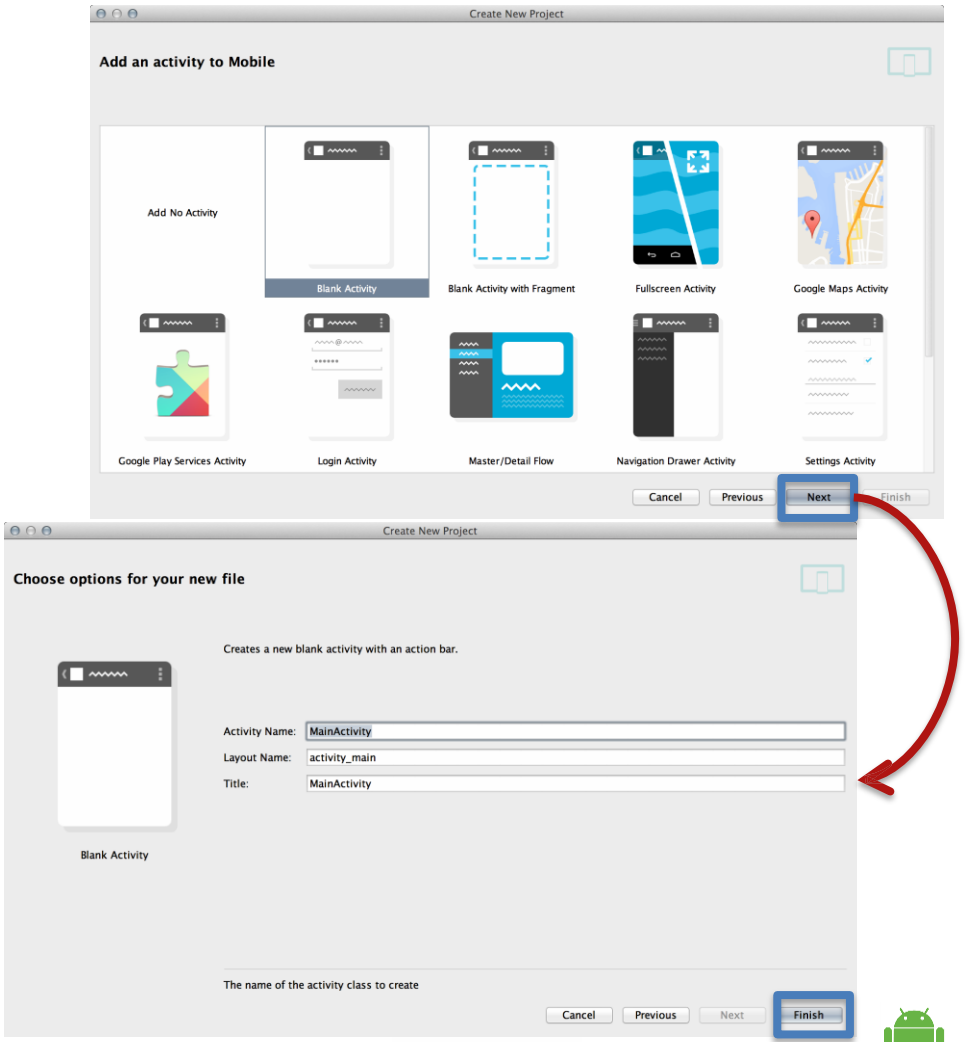
Creating Android App (1)

- Creating Android app project in Android Studio:
- Go to *File→New Project*
- Enter app, project name
- Choose package name using "reverse URL" notation, e.g., edu.osu.myapp
- Select APIs for app, then click Next



Creating Android App (2)

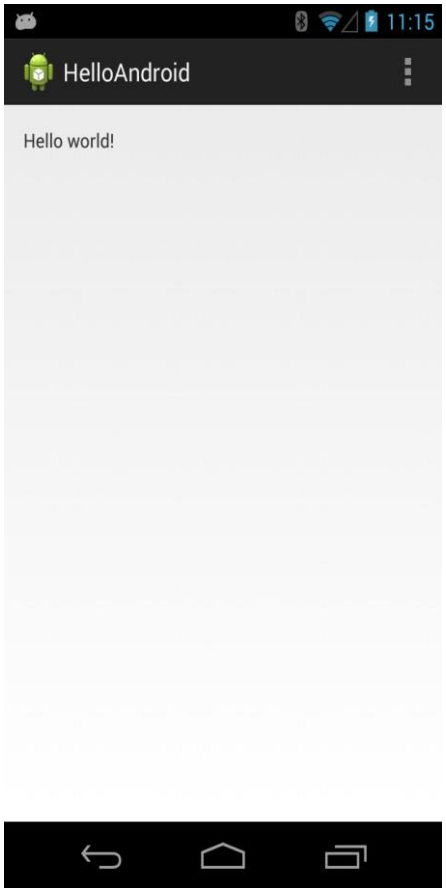
- Determine what kind of Activity to create; then click Next
- We'll choose a Blank Activity for simplicity
- Enter information about your Activity, then click Finish
- This creates a "Hello World" app

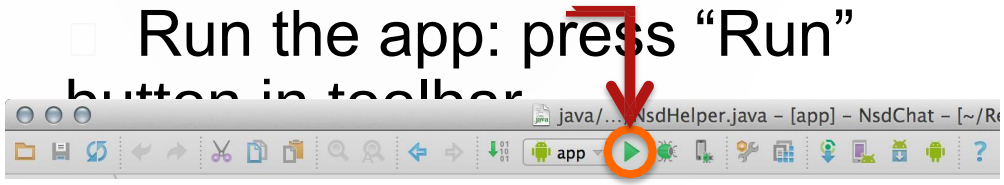


Android

Deploying the App

- Two choices for deployment:
  - Real Android device
  - Android virtual device
- Plug in your real device; otherwise, create an Android virtual device
- Emulator is slow. Try Intel accelerated version, or perhaps http://www.genymotion.com/
- Run the app: press "Run" button in toolbar





Underlying GUI Code

res/layout/activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"  android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"  tools:context=".MainActivity" >

<TextView
android:layout_width="wrap_content"  android:layout_height="wrap_content"
android:text="@string/hello_world" />
</RelativeLayout>
```

Underlying Source Code

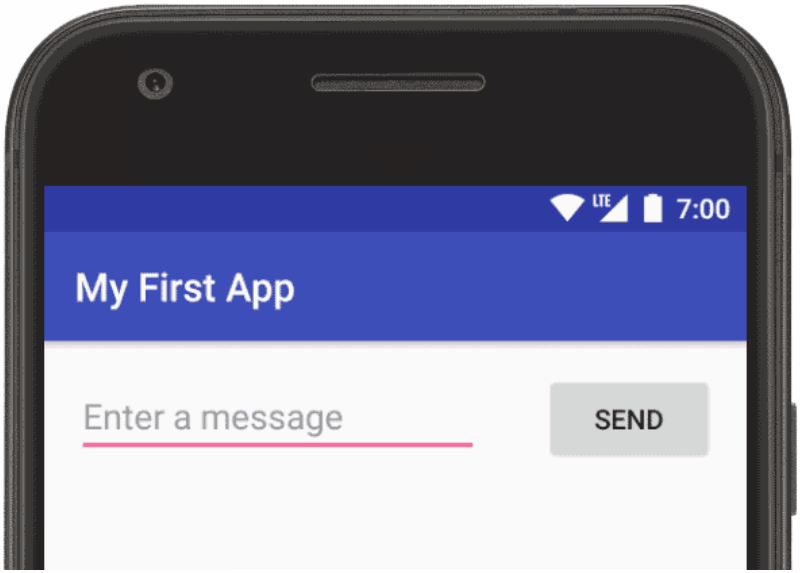
```
package edu.osu.helloandroid;  import android.os.Bundle;  import android.app.Activity;  import
android.view.Menu;
public class MainActivity extends Activity
{
@Override
protected void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);  setContentView(R.layout.activity_main);
}

@Override
public boolean onCreateOptionsMenu(Menu menu)
{
// Inflate the menu; this adds items to the action bar if it is present.
getMenuInflater().inflate(R.menu.main, menu);  return true;
}
}
```


Hardware-oriented Features

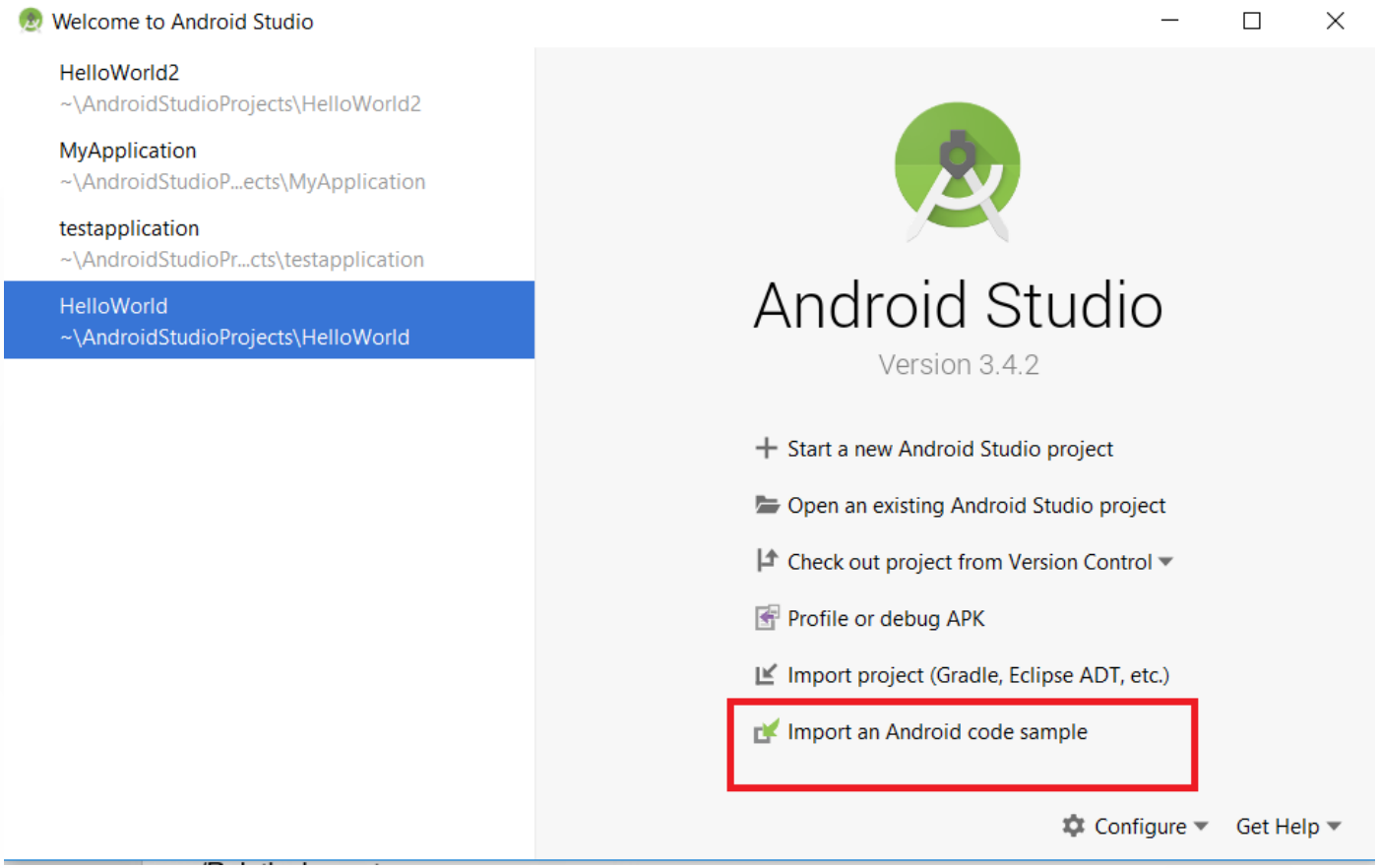
Feature	Description
Camera	A class that enables your application to interact with the camera to snap a photo, acquire images for a preview screen, and modify parameters used to govern how the camera operates.
Sensor	Class representing a sensor. Use getSensorList(int) to get the list of available Sensors.
SensorManager	A class that permits access to the sensors available within the Android platform.
SensorEventListener	An interface used for receiving notifications from the SensorManager when sensor values have changed. An application implements this interface to monitor one or more sensors available in the hardware.
SensorEvent	This class represents a sensor event and holds information such as the sensor type (e.g., accelerometer, orientation, etc.), the time-stamp, accuracy and of course the sensor's data.
MediaRecorder	A class, used to record media samples, that can be useful for recording audio activity within a specific location (such as a baby nursery). Audio clippings can also be analyzed for identification purposes in an access-control or security application. For example, it could be helpful to open the door to your time-share with your voice, rather than having to meet with the realtor to get a key.
GeomagneticField	This class is used to estimated estimate magnetic field at a given point on Earth, and in particular, to compute the magnetic declination from true north.
FaceDetector	A class that permits basic recognition of a person's face as contained in a bitmap. Using this as a device lock means no more passwords to remember — biometrics capability on a cell phone.

Next Walk Through Androids Tutorial

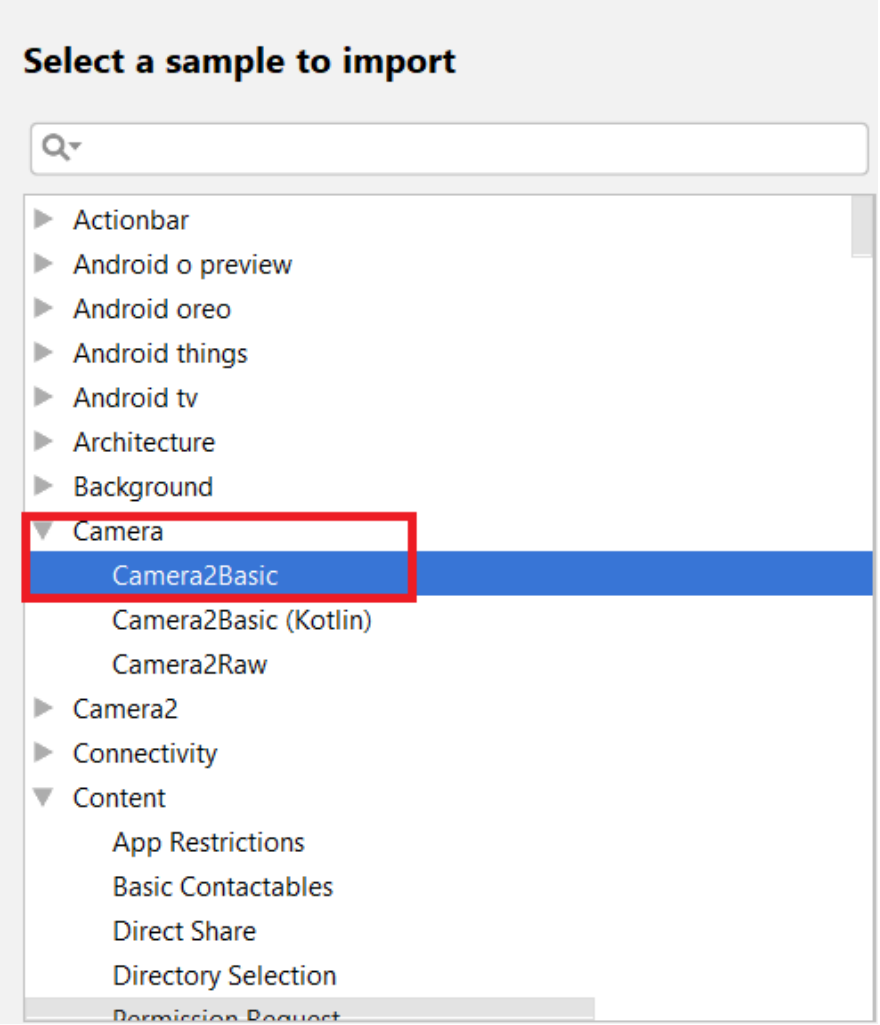
https://developer.android.com/training/basics/firstapp



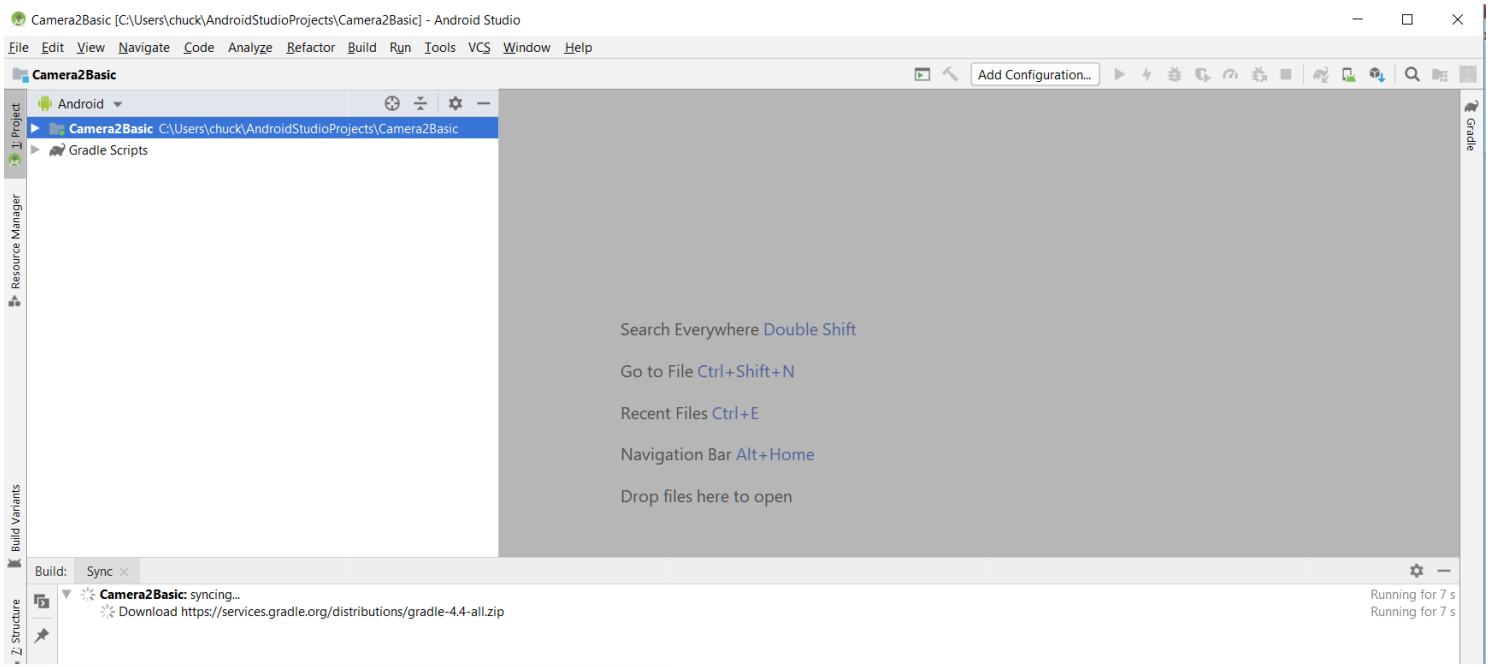
Examine a Sample App



Examine a Sample App



Examine a Sample App



Android

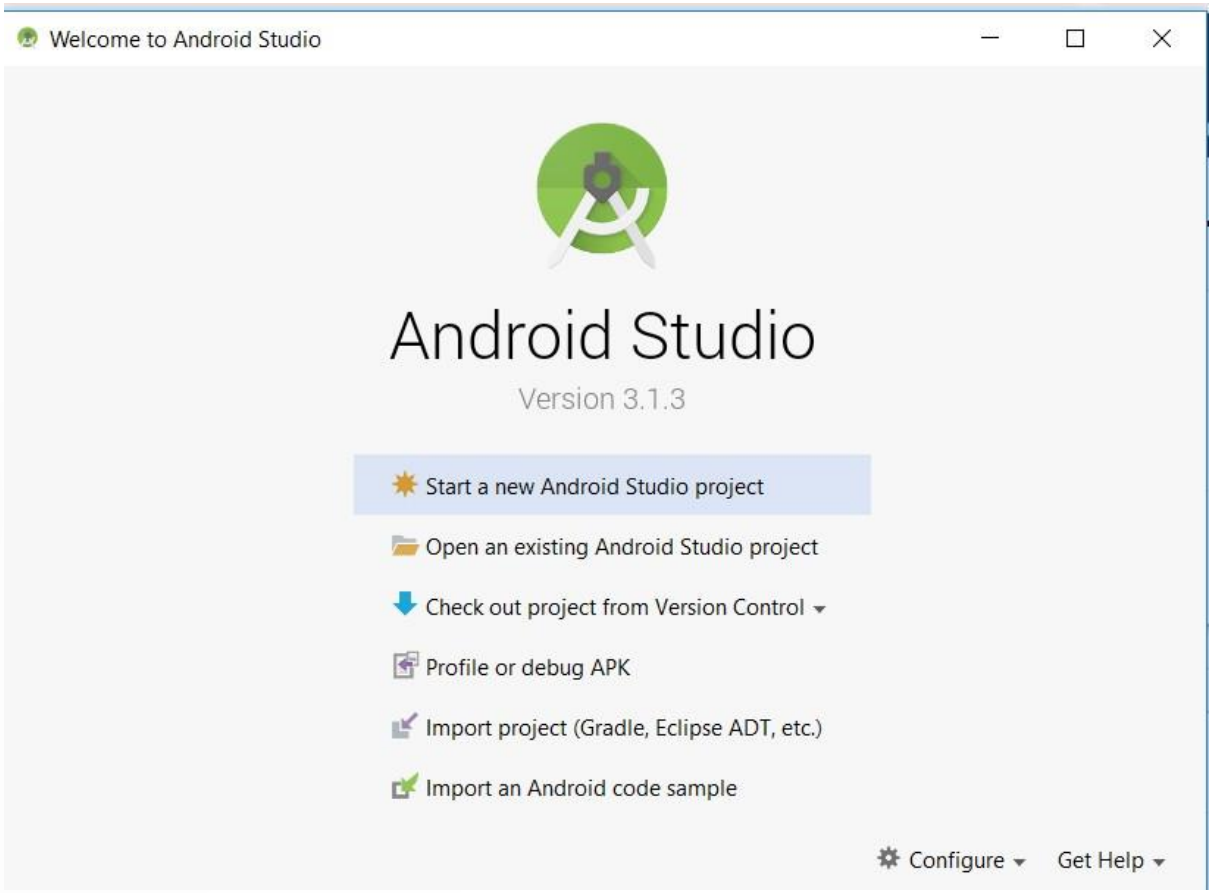
Feature	Description
Camera	A class that enables your application to interact with the camera to snap a photo, acquire images for a preview screen, and modify parameters used to govern how the camera operates.
Sensor	Class representing a sensor. Use getSensorList(int) to get the list of available Sensors.
SensorManager	A class that permits access to the sensors available within the Android platform.
SensorEventListener	An interface used for receiving notifications from the SensorManager when sensor values have changed. An application implements this interface to monitor one or more sensors available in the hardware.
SensorEvent	This class represents a sensor event and holds information such as the sensor type (e.g., accelerometer, orientation, etc.), the time-stamp, accuracy and of course the sensor's data.
MediaRecorder	A class, used to record media samples, that can be useful for recording audio activity within a specific location (such as a baby nursery). Audio clippings can also be analyzed for identification purposes in an access-control or security application. For example, it could be helpful to open the door to your time-share with your voice, rather than having to meet with the realtor to get a key.
GeomagneticField	This class is used to estimated estimate magnetic field at a given point on Earth, and in particular, to compute the magnetic declination from true north.
FaceDetector	A class that permits basic recognition of a person's face as contained in a bitmap. Using this as a device lock means no more passwords to remember — biometrics capability on a cell phone.

Hardware-oriented Features

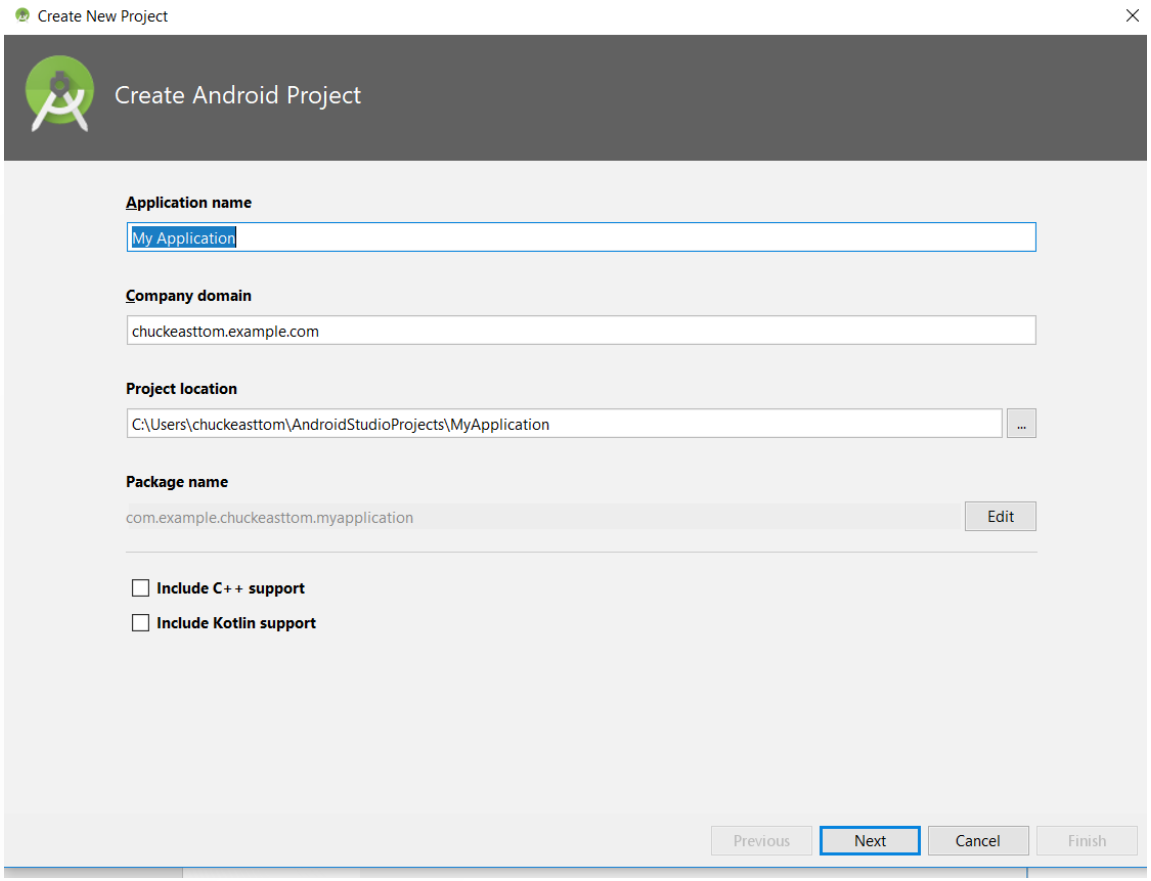
- Sensor type (Sensor class)
  - Orientation, accelerometer, light, magnetic field, proximity, temperature, etc.
- Sampling rate
  - Fastest, game, normal, user interface.
  - When an application requests a specific sampling rate, it is really only a hint, or suggestion, to the sensor subsystem. There is no guarantee of a particular rate being available.
- Accuracy
  - High, low, medium, unreliable.

Sensor and SensorManager

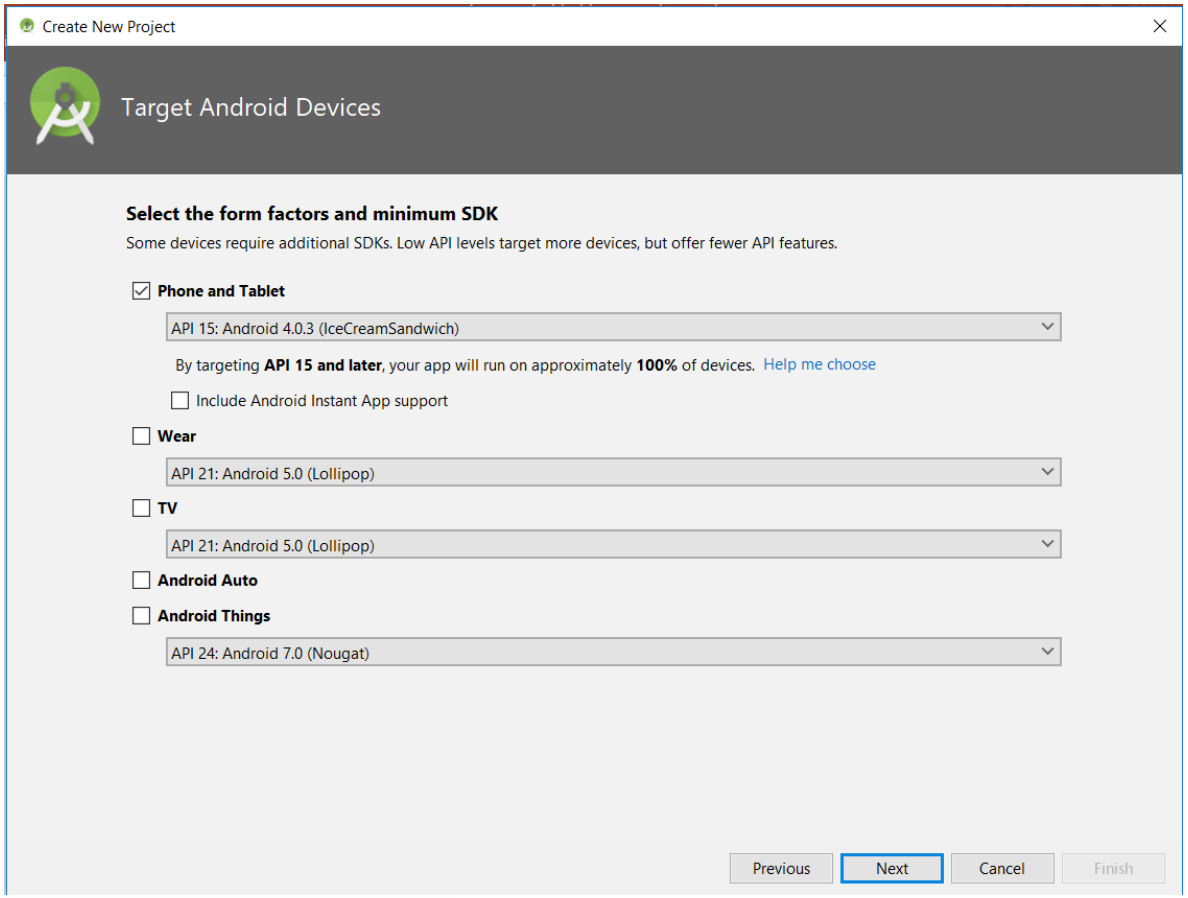
Your First App



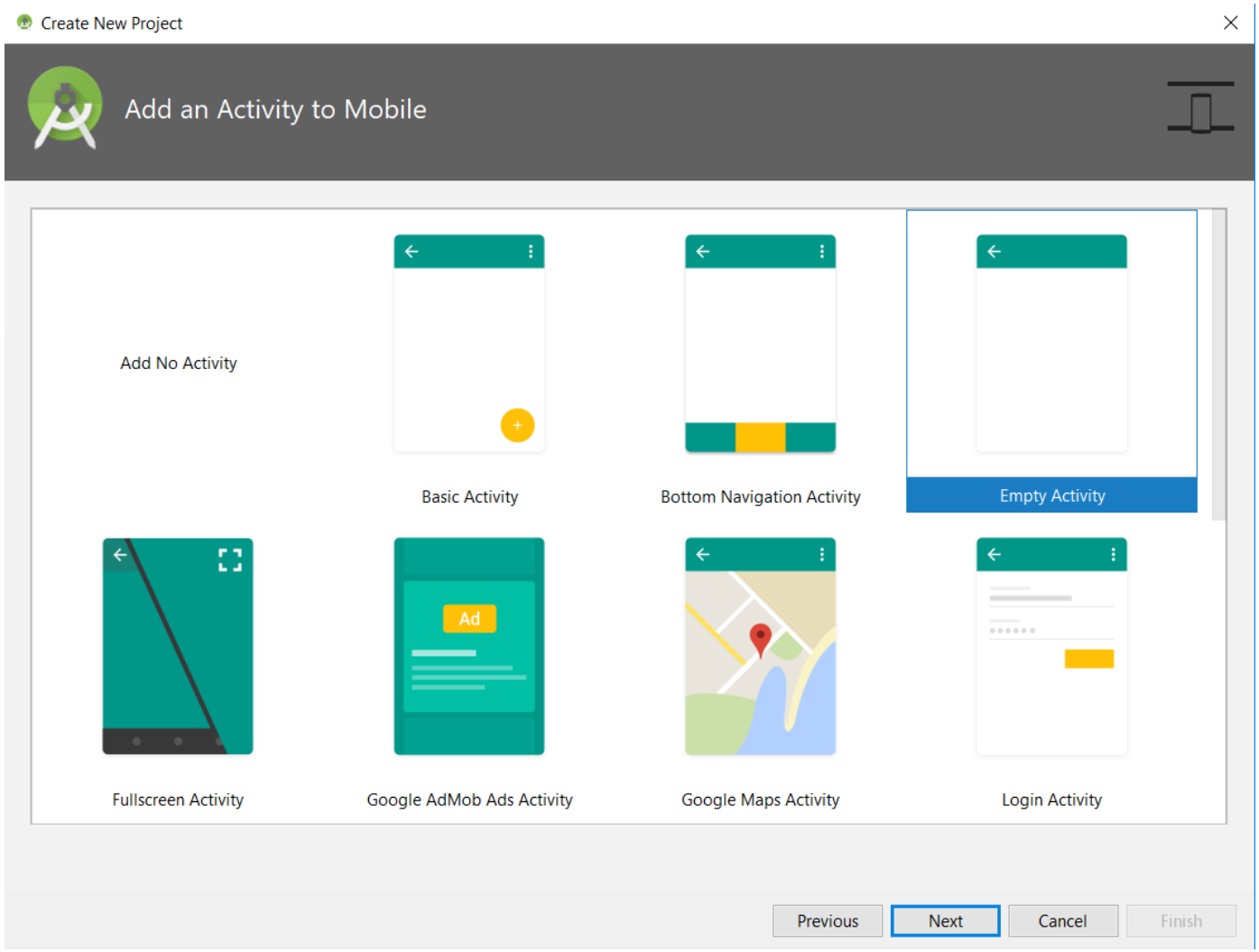
Your First App



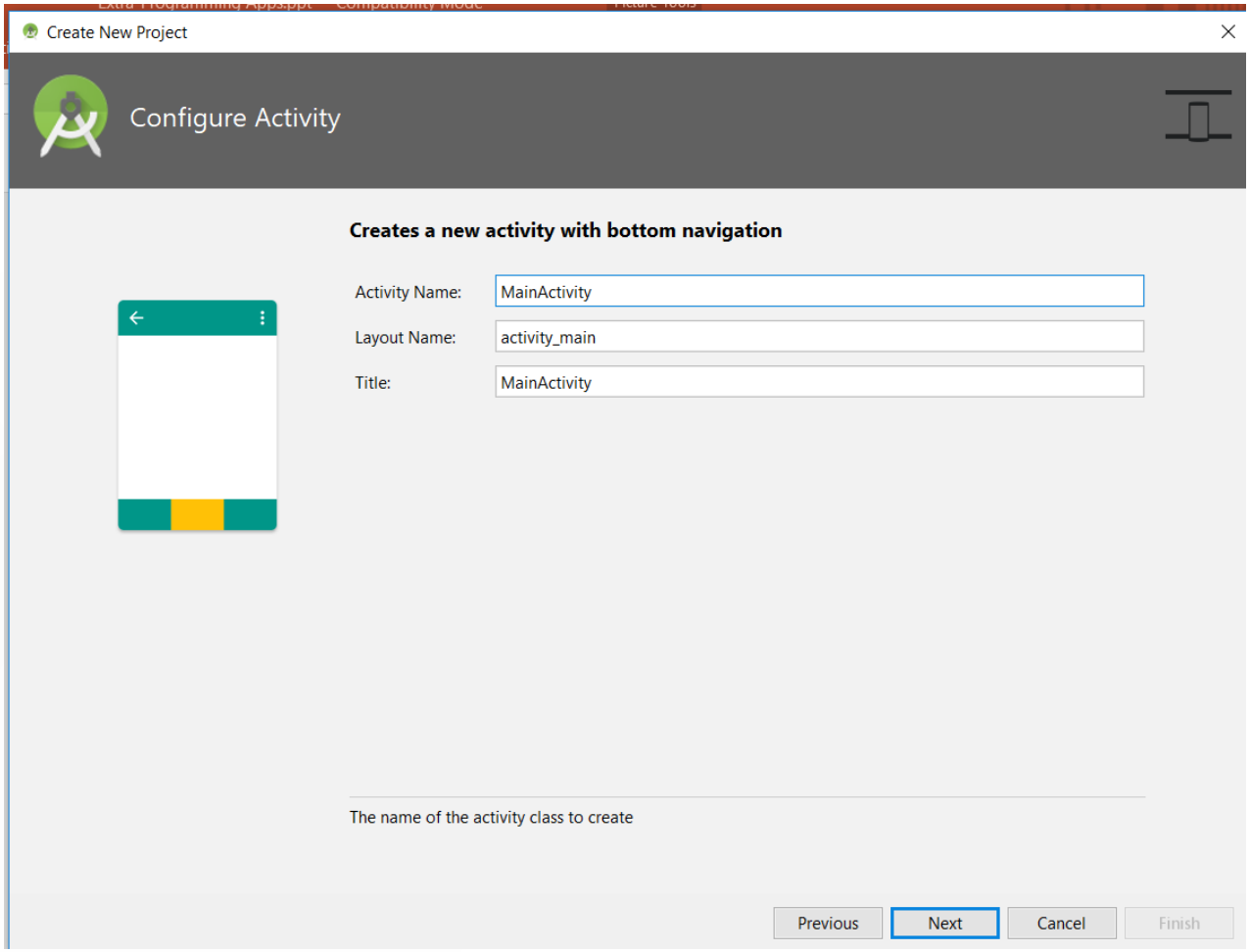
Your First App



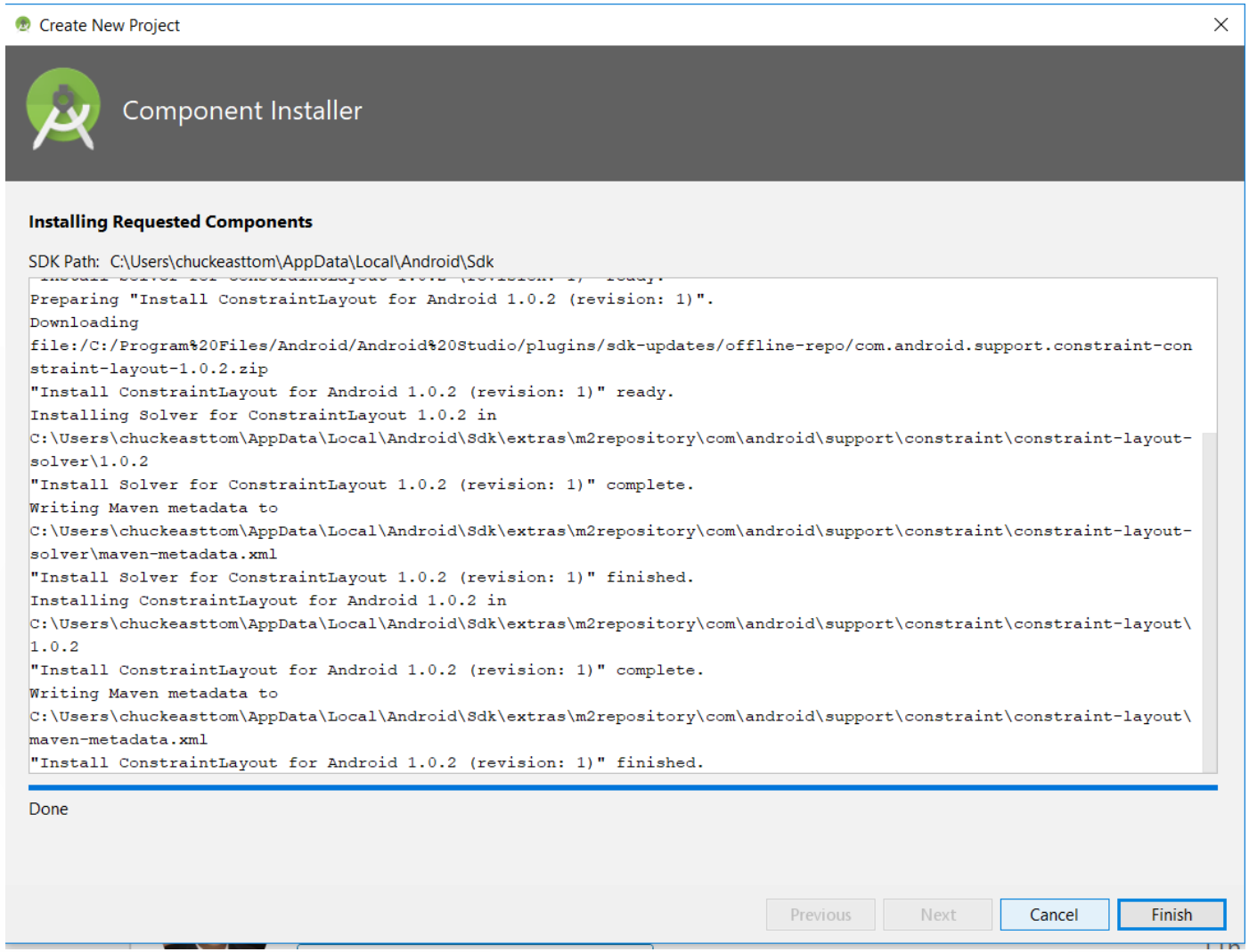
Your First App



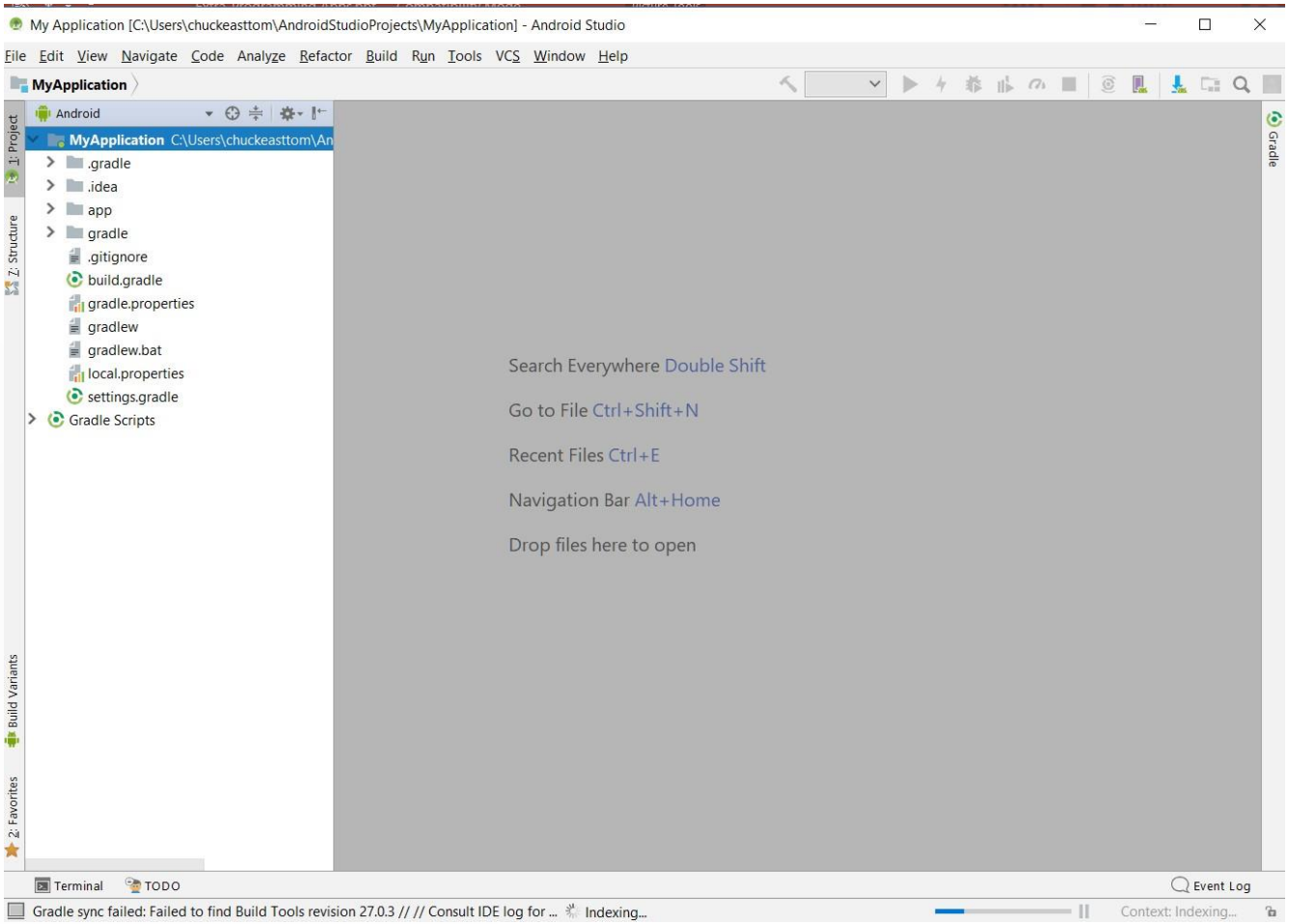
Your First App



Your First App

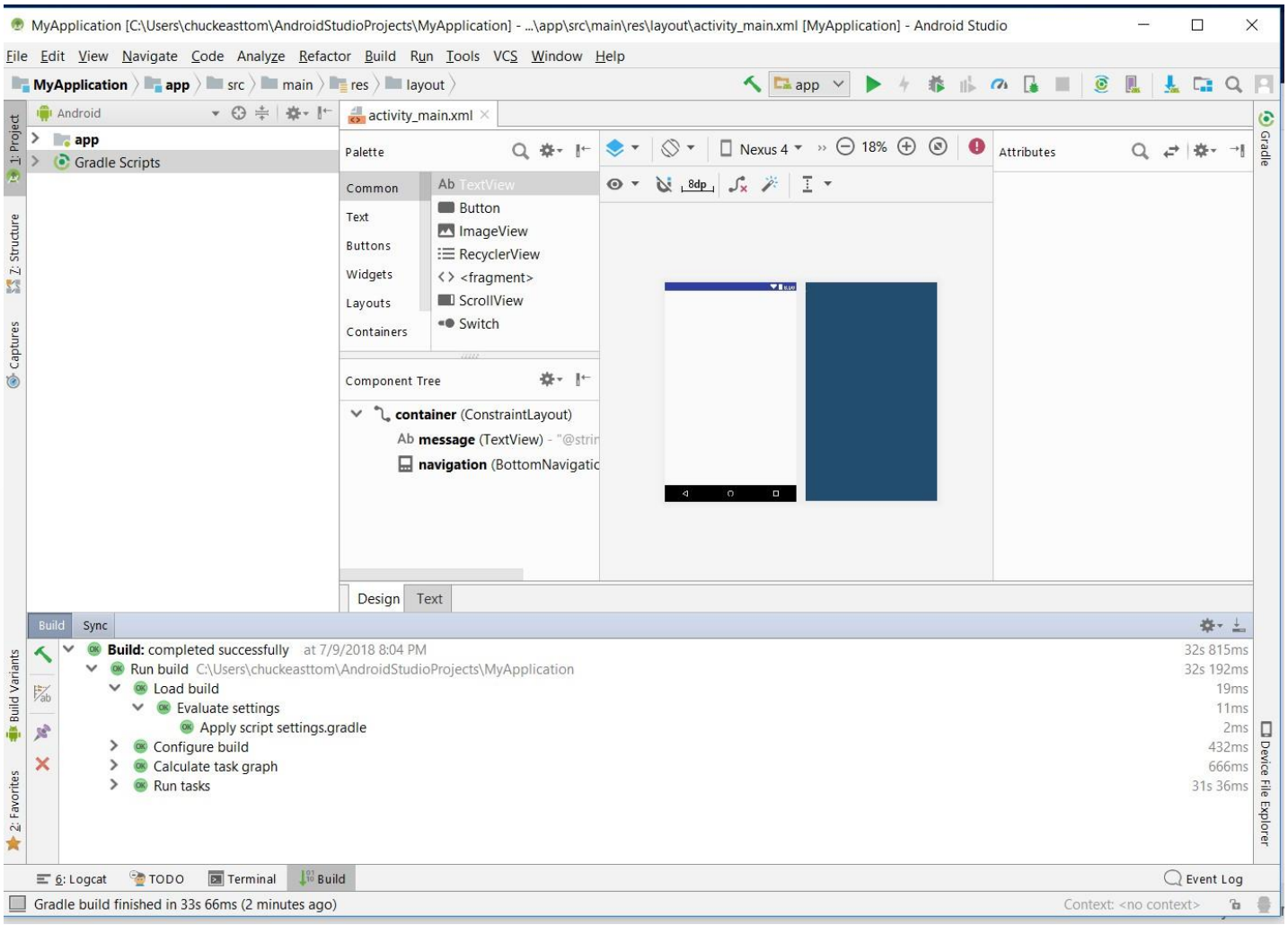


Your First App



Android

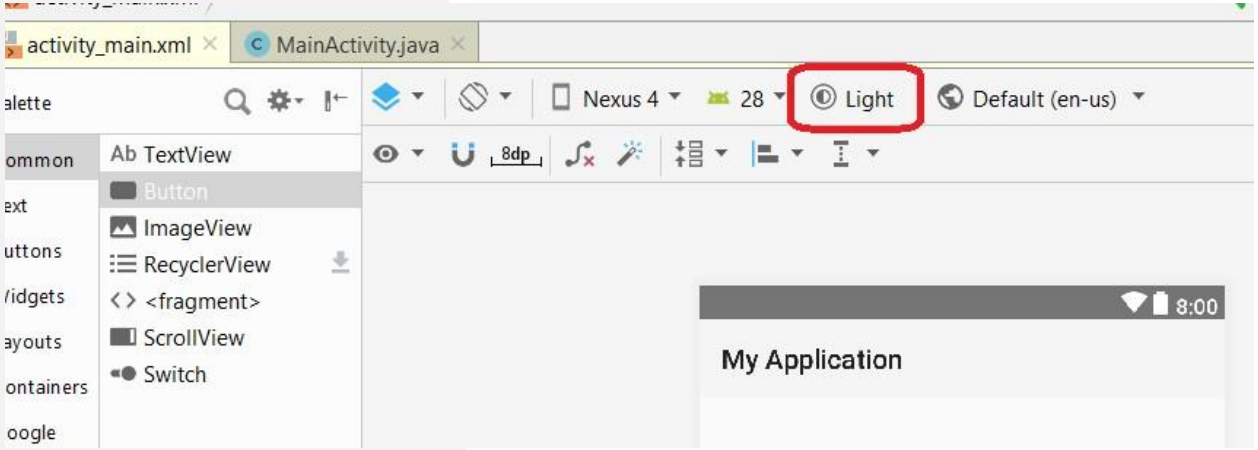
Your First App



Android

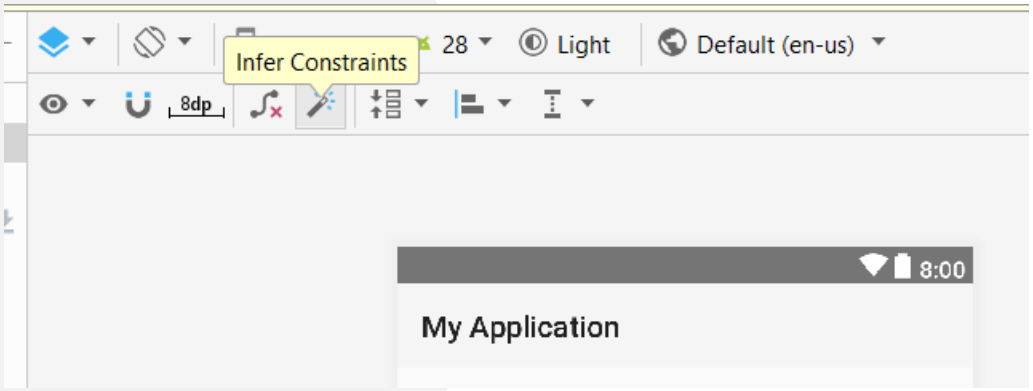
Design View

- NOTE: If you cannot drag and drop components to the design view, then add a theme (any theme)



Infer Constraints

- Note: You will get warnings frequently regarding layout constraints. It is easier just to use ‘infer constraints’



Create code for a button click

```
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);

        button.setOnClickListener(new View.OnClickListener()
        {

            public void onClick(View v)
            {
                // Code here executes on main thread after user presses button

            }

        });
    }
}
```

Add some code

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        final TextView text = (TextView) findViewById(R.id.textView);

        button.setOnClickListener(new View.OnClickListener()
        {

            public void onClick(View v)
            {
                text.setText("QHHH");
            }

        });
    }
}
```

Geolocation App

Set the Manifest

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

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />

    <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>
    </application>

</manifest>
```


Geolocation App - 2

Request Permissions

```
if (ContextCompat.checkSelfPermission(getApplicationContext(), android.Manifest.permission.ACCESS_
FINE_LOCATION) != PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(getAppli
cationContext(), android.Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_
GRANTED) {

      ActivityCompat.requestPermissions(this, new String[]{android.Manifest.permission.ACCESS_FIN
E_LOCATION, android.Manifest.permission.ACCESS_COARSE_LOCATION}, 101);

}
```


Geolocation App - 3

Code in Context

```
import android.widget.TextView;
import android.widget.Toast;

import java.util.List;
import java.util.Locale;

public class MainActivity extends AppCompatActivity implements LocationListener {

    Button getLocationBtn;
    TextView locationText;

    LocationManager locationManager;

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

        getLocationBtn = (Button)findViewById(R.id.getLocationBtn);
        locationText = (TextView)findViewById(R.id.locationText);


        if (ContextCompat.checkSelfPermission(getApplicationContext(), android.Manifest.permissio
n.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission
(getApplicationContext(), android.Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PE
RMISSION_GRANTED) {

            ActivityCompat.requestPermissions(this, new String[]{android.Manifest.permission.ACCES
S_FINE_LOCATION, android.Manifest.permission.ACCESS_COARSE_LOCATION}, 101);
```


Geolocation App - 4

Code in Context

```
    void getLocation() {
        try {
            locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
            locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 5000, 5, thi
s);
        }
        catch(SecurityException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onLocationChanged(Location location) {
        locationText.setText("Latitude: " + location.getLatitude() + "\n Longitude: " + location.g
etLongitude());

        try {
            Geocoder geocoder = new Geocoder(this, Locale.getDefault());
            List<Address> addresses = geocoder.getFromLocation(location.getLatitude(), location.ge
tLongitude(), 1);
            locationText.setText(locationText.getText() + "\n"+addresses.get(0).getAddressLine
(0)+", "+
                                 addresses.get(0).getAddressLine(1)+", "+addresses.get(0).getAddressLin
e(2));
        }catch(Exception e)
        {

        }
```

https://androstock.com/tutorials/getting-current-location-latitude-longitude-country-android-android-studio.html

Get Contacts - 1

Manifest Permissions

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

```
import android.provider.ContactsContract;
```

https://developer.android.com/training/contacts-provider/retrieve-names

Get Contacts - 2

Code View

```
...
/*
 * Defines an array that contains column names to move from
 * the Cursor to the ListView.
 */
@SuppressLint("InlinedApi")
private final static String[] FROM_COLUMNS = {
        Build.VERSION.SDK_INT
                >= Build.VERSION_CODES.HONEYCOMB ?
                ContactsContract.Contacts.DISPLAY_NAME_PRIMARY :
                ContactsContract.Contacts.DISPLAY_NAME
};
/*
 * Defines an array that contains resource ids for the layout views
 * that get the Cursor column contents. The id is pre-defined in
 * the Android framework, so it is prefaced with "android.R.id"
 */
private final static int[] TO_IDS = {
       android.R.id.text1
};
// Define global mutable variables
// Define a ListView object
ListView contactsList;
// Define variables for the contact the user selects
// The contact's _ID value
long contactId;
// The contact's LOOKUP_KEY
String contactKey;
// A content URI for the selected contact
Uri contactUri;
// An adapter that binds the result Cursor to the ListView
private SimpleCursorAdapter cursorAdapter;
...
```


Activities

- An *activity* is the entry point for interacting with the user. It represents a single screen with a user interface. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails

Services

- A *service* is a general-purpose entry point for keeping an app running in the background for all kinds of reasons. It is a component that runs in the background to perform long-running operations or to perform work for remote processes.

Broadcast Receiver

- A *broadcast receiver* is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements. Because broadcast receivers are another well-defined entry into the app, the system can deliver broadcasts even to apps that aren't currently running.

Binder

- Binder driver, which implements inter-process communication (IPC).

Binder

Perhaps one of the most important additions to Android's Linux kernel was a driver known as *Binder*. Binder is an IPC mechanism based on a modified version of OpenBinder, originally developed by Be, Inc., and later Palm, Inc. Android's Binder is relatively small (approximately 4,000 lines of source code across two files), but is pivotal to much of Android's

Drake, Joshua J.; Lanier, Zach; Mulliner, Collin; Oliva Fora, Pau; Ridley, Stephen A.; Wicherski, Georg. Android Hacker's Handbook

Zygote

Zygote One of the first processes started when an Android device boots is the Zygote process. Zygote, in turn, is responsible for starting additional services and loading libraries used by the Android Framework. The Zygote process then acts as the loader for each Dalvik process by creating a copy of itself, or forking.
This optimization prevents having to repeat the expensive process of loading the Android Framework and its dependencies when starting Dalvik processes (including apps). As a result, core libraries, core classes, and their corresponding heap structures are shared across instances of the DalvikVM. Zygote's second order of business is starting the system_server process.

Drake, Joshua J.; Lanier, Zach; Mulliner, Collin; Oliva Fora, Pau; Ridley, Stephen A.; Wicherski, Georg. Android Hacker's Handbook

Application Fundamentals

- Activities
  - application presentation layer
- Services
  - invisible components, update data sources, visible activities, trigger notifications
  - perform regular processing even when app is not active or invisible
- Content Providers
  - shareable data store
- Intents
  - message passing framework
  - broadcase messages system wide, for an action to be performed
- Broadcast receivers
  - consume intent broadcasts
  - lets app listen for intents matching a specific criteria like location
- Notifications
  - Toast notification
  - Status Bar Notification
  - Dialog notification

Android S/W Stack –App Framework

Feature	Role
View System	Used to build an application, including lists, grids, text boxes, buttons, and embedded web browser
Content Provider	Enabling applications to access data from other applications or to share their own data
Resource Manager	Providing access to non-code resources (localized string , graphics, and layout files)
Notification Manager	Enabling all applications to display customer alerts in the status bar
Activity Manager	Managing the lifecycle of applications and providing a common navigation backstack

Basics

- The file must have the same name as the public class in it
- File must end in .java extension
- Java is CasE SensITive.

Primitive Data Types

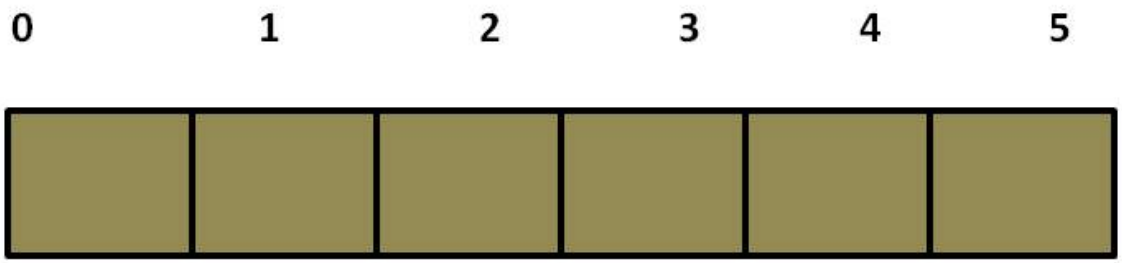
- byte – 8 bit integer with values -127 to +127
- short – 16 bit integer with values - 32,768 to + 32,768
- int– 32 bit integer with values - 2,147,483,648 to + 2,147,483,648
- long – 64 bit integer with values - 9,223,372,036,854,775,808 to + 9,223,372,036,854,775,808
- float – 32 bit decimal number.
- double - 64 bit decimal number
- boolean – true or false
- char – 16 bit unicode character

Reference Data Types

- Array
- Class
- Interface

Arrays

- int[] anArray;
- Elements (like what is at anArray[3])
- Index (the number designating what cell we are at).
- Zero based



Array length is 6

Arrays continued

- int i[]; /* one dimension array */
- int j[][]; /* two dimension array */

- Array1 = new int[5]; //previously declared, now created
- int Array2[] = new int[5]; //declaration and allocation at same time
- 2D array int m[][] = {{2,3,4}, {4,5,6}, {1,1,1}}; // two dimensional array

String

- String str = "abc";
- String str1=”Hello”;
- String str2=”Hello”;

Naming Conventions

- Use the first letter with an underscore or use the first three letters such as:
  - An integer might be i_AccountNum or intAccountNum
  - A boolean might be b_IsValid or boolIsValid

Variables

- Java Supports dynamic initialization
  - int interest= balance *.02
- Variable Scope
  - Block, function, class, namespace

Reserved Words

- abstract
- const
- finally
- return
- throws
- assert
- continue
- interface
- transient
- default
- for
- static
- try
- break
- do
- goto
- native
- strictfp
- void
- if
- new
- super
- volatile
- case
- else
- implements
- package
- switch
- while
- catch
- enum
- private
- synchronized
- extends
- import
- protected
- class
- this
- instance of
- public
- throw

Basic Operators

- +, -, *, / add, subtract, multiply, divide
- % modulus or remainder
- ++ increment by one
- -- decrement by one
- ! Logical not
- && Logical and
- || Logical OR
- == is it equal
- = make it equal
- != not equal
- > greater than
- < less than
- >= greater than or equal
- <= less than or equal
- += add then assign
- -= subtract then assign

Expressions

- A statement is a single line of code that performs some action. Such as
  - int x = 0;
  - accountbalance = balance + interest;
  - balance+= interest;
  - All statements end with a semi colon ;
- A statement can be declarative or execution

Calling a function is a statement, and ends with a semi colon

- functionname(parameters);
- computeinterest(55);

The Basic HELLO World

```
class HelloWorldApp
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

Some Basics

- Functions are
  - Access level, return type, name, parameters
  - {} define code blocks (functions, blocks, etc.)

If Statements

```
if(some condition exists)
{
    Do this action
}

if( age <12)
{
    compute_full_price();
}

if(age < 12)
   compute_full_price();
```


For Loops

```
for (initialization; termination; increment)
{
   statement(s)
}

for (int j = 0; j< count; j++)
{
      computeinterest(banlance[j]);
}
```

While loops

```
while (someconditionexists)
{
   // execute code
}

while (totalItems< 1000)
{
    addNewItem();
}
```

Basic Example Factorial

```
public class FactorialExample
{

   public static void main(String[] args)
   {
       int number = 5;
       int factorial = number ;

        for(int i =(number - 1); i > 1; i--)
       {
         factorial = factorial * i;
        }

        System.out.println("Factorial of a number is " + factorial);
   }// end of main
}// end of class FactorialExample
```


Handling Exceptions

```
try
{
// code you want to have happen
}
catch (ExceptionType name)
{
 //what to do in case of an error
}
catch (ExceptionType name)
{
//what to do in case of a different error
}
finally
{
  // stuff that happens no matter what
}
```


Exception example

```
public class ExceptionExample{
      public static void main(String ar[])
      {
            try
           {
                  System.out.println(10/0);
            }
            catch(ArithmeticException e)
           {
                  System.out.println("caught"); // caught
                 // for see exception information these three methods

                  //to print exception stack this method
                   e.printStackTrace(); //don't put in the inside sop.

                  //to print exception name and message use this method
                  System.out.println(e.toString());

                  // to print only message
                    System.out.println(e.getMessage());

            }
      }
```


Common Mistakes

- Incorrect brackets
- Missing semi colon
- Case sensitivity

Classes and Objects

- The ***class*** is the unit of programming
- A Java program is a ***collection of classes***
  - Each class definition (usually) in its own `.java` file
  - *The file name must match the class name*
- A class describes ***objects (instances)***
  - Describes their common characteristics: is a *blueprint*
  - Thus all the instances have these same characteristics
- These characteristics are:
  - ***Data fields*** for each object
  - ***Methods*** (operations) that do work on the objects

More With Java

Comments

- Comments are ignored and are treated as white space
- They should be written to enhance readability
  - Explain what a piece of code does (its *interface*)
  - Explain any special tricks, limitations, …
- Java has three comment formats:
  - **// comment to end of line**
  - **/*comment until closing */**
  - **/** API specification comment */**

Variable Declarations

- The syntax of a variable declaration is

```
int total;
```

- For example

```
long total, count, sum;
```

- Multiple variables can be declared on the same line

```
int total = 0, count = 20;
double unitPrice = 57.25;
```

- in the declaration

Variable

```
public class DeclarationExample {  public
    static void main (String[]                    args) {
        int weeks = 14;
        long numberOfStudents = 120;
        double averageFinalGrade = 78.6;
        System.out.println(weeks);
        System.out.println(numberOfStudents);
        System.out.println(averageFinalGrade);
        }
        }
```

More Variable

```
double pi, conversionRate, temprature;
long salary;  boolean isOn;  char c;


pi = 3.14159;
isOn = false;
c = 'A';
salary = 34000;  isOn = true;
```

Arithmetic Operators

- An ***operator*** is a mapping that maps one or more values to a single value:
- Binary Operators:

a + b	adds a and b
a - b	subtracts b from a
a * b	multiplies a and b
a / b	divides a by b
a % b	the reminder of divining a by b

- Unary Operator:

-a	The negation of a

Pounds to Kg

```
public class PoundsToKg {
public static void main(String[] args){  double
weightInPounds = 200.0;
final double KILOS_IN_POUND = 0.455;
double weightInKg;

weightInKg = weightInPounds   KILOS*_IN_POUND ;
System.out.println(weightInKg);
}
}
```

Pounds to Kg

```
public class PoundsToKg2 {
public static void main(String[] args){  final double
KILOS_IN_POUND = 0.455;  System.out.println(200.0
    KILOS_IN_POUND);              *
}
}
```

Transmitting SMS messages across the network -- Programming Tutorial 3

Intent and IntentFilter

- Intents request for an action to be performed and supports interaction among the Android components.
  - For an activity it conveys a request to present an image to the user
  - For broadcast receivers, the Intent object names the action being announced.
- Intent Filter Registers Activities, Services and Broadcast Receivers(as being capable of performing an action on a set of data).

SMS Sending

- STEP 1
  - In the AndroidManifest.xml file, add the two permissions - SEND_SMS and RECEIVE_SMS.
- STEP 2
  - In the main.xml, add Text view to display "Enter the phone number of recipient“ and "Message"
  - EditText with id txtPhoneNo and txtMessage
  - Add the button ID "Send SMS“

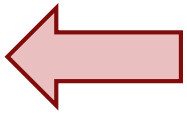
- Step 3 Import Classes and Interfaces

```
import android.app.Activity;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
```

SMS Sending

Step 4 Write the SMS class

```
public class SMS extends Activity {
  Button btnSendSMS;
  EditText txtPhoneNo;
  EditText txtMessage;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    btnSendSMS = (Button) findViewById(R.id.btnSendSMS);
    txtPhoneNo = (EditText) findViewById(R.id.txtPhoneNo);
    txtMessage = (EditText) findViewById(R.id.txtMessage);

    btnSendSMS.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        String phoneNo = txtPhoneNo.getText().toString();
        String message = txtMessage.getText().toString();

        if (phoneNo.length()>0 && message.length()>0)
          sendSMS(phoneNo, message);
        else
          Toast.makeText(getBaseContext(),
            "Please enter both phone number and message.",
            Toast.LENGTH_SHORT).show();
      }
    });
  }
}
```



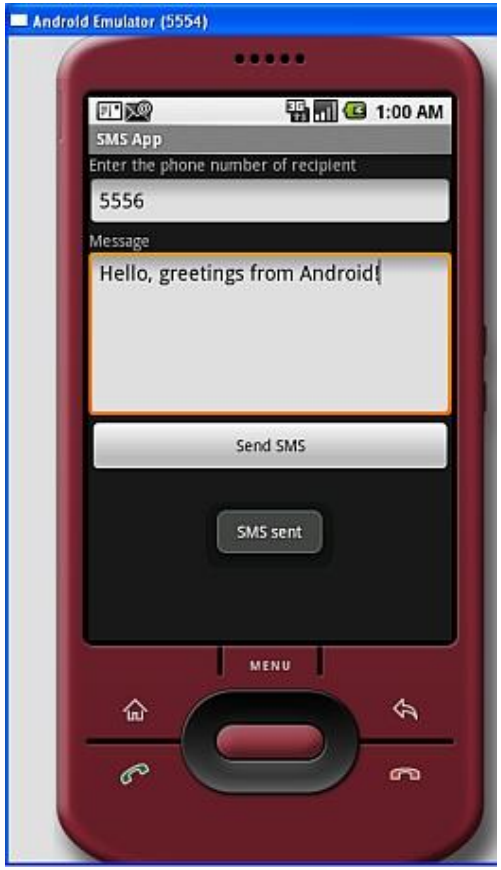
Input from the user (i.e., the phone no, text message and sendSMS is implemented).

SMS Sending

- Step 5
  - To send an SMS message, you use the SmsManager class. And to instantiate this class call getDefault() static method.
  - The sendTextMessage() method sends the SMS message with a PendingIntent.
  - The PendingIntent object is used to identify a target to invoke at a later time.

```
private void sendSMS(String phoneNumber, String message) {
  PendingIntent pi = PendingIntent.getActivity(this, 0,
    new Intent(this, SMS.class), 0);
  SmsManager sms = SmsManager.getDefault();
  sms.sendTextMessage(phoneNumber, null, message, pi, null);
}
```

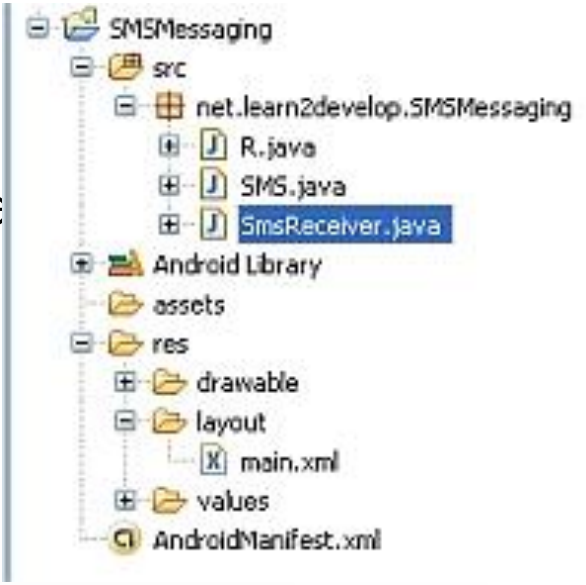
SMS Sending


Android Emulator (5554)
1:00 AM
SMS App
Enter the phone number of recipient
5556
Message
Hello, greetings from Android!
Send SMS
SMS sent
MENU

SMS Sending

Receiving SMS

- Ste



Receiving SMS

- Step 2
  - In the AndroidManifest.xml file add the <receiver> element so that incoming SMS messages can be intercepted by the SmsReceiver class.

```
<receiver android:name=".SmsReceiver">
        <intent-filter>
            <action android:name=
"android.provider.Telephony.SMS_RECEIVED" />
        </intent-filter>
    </receiver>
```

Receiving SMS

- Step 3

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.telephony.SmsMessage;
import android.widget.Toast;
```

Step 4

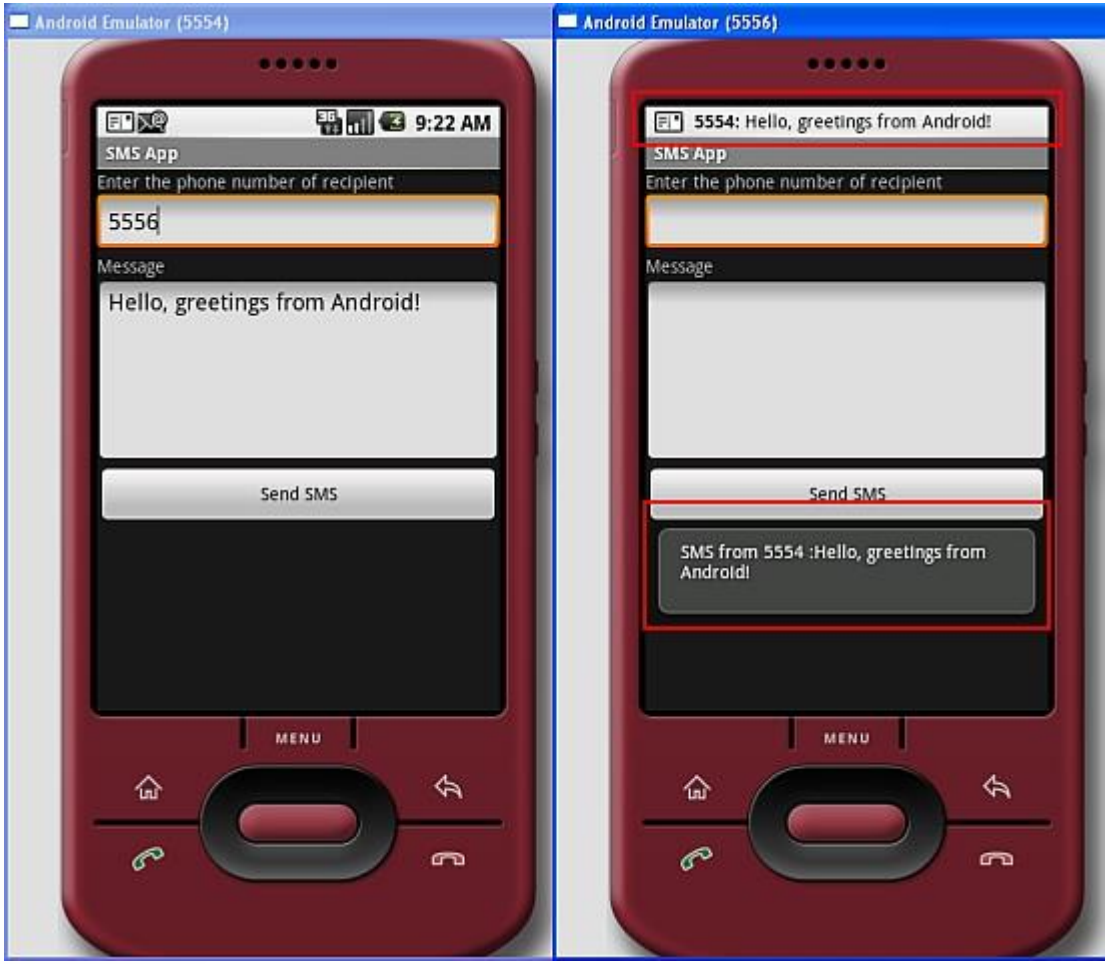
```
public class SmsReceiver extends BroadcastReceiver  {

  @Override

  public void onReceive(Context context, Intent intent) {

    //---get the SMS message passed in---

    Bundle bundle = intent.getExtras();

    SmsMessage[] msgs = null;

    String str = "";


    if (bundle != null){

      //---retrieve the SMS message received---

      Object[] pdus = (Object[]) bundle.get("pdus");

      msgs = new SmsMessage[pdus.length];


      for (int i=0; i<msgs.length; i++) {

        msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);

        str += "SMS from " + msgs[i].getOriginatingAddress();

        str += " :";

        str += msgs[i].getMessageBody().toString();

        str += "\n";

      }

      //---display the new SMS message---

      Toast.makeText(context, str, Toast.LENGTH_SHORT).show();

    }

  }

}
```

In the SmsReceiver class, extend the BroadcastReceiver class and override the onReceive() method. The message is attached to the Intent

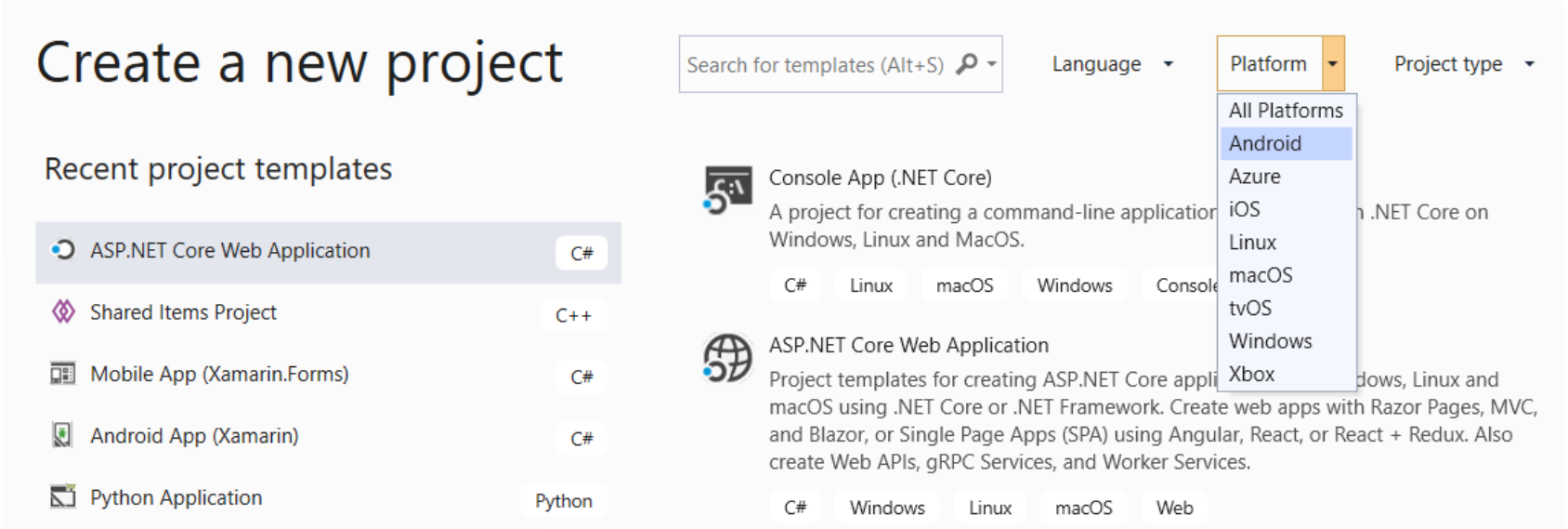
The messages are stored in a object array PDU format. To extract each message, you use the static createFromPdu() method from the SmsMessage class. The SMS message is then displayed using the Toast class

Receiving SMS

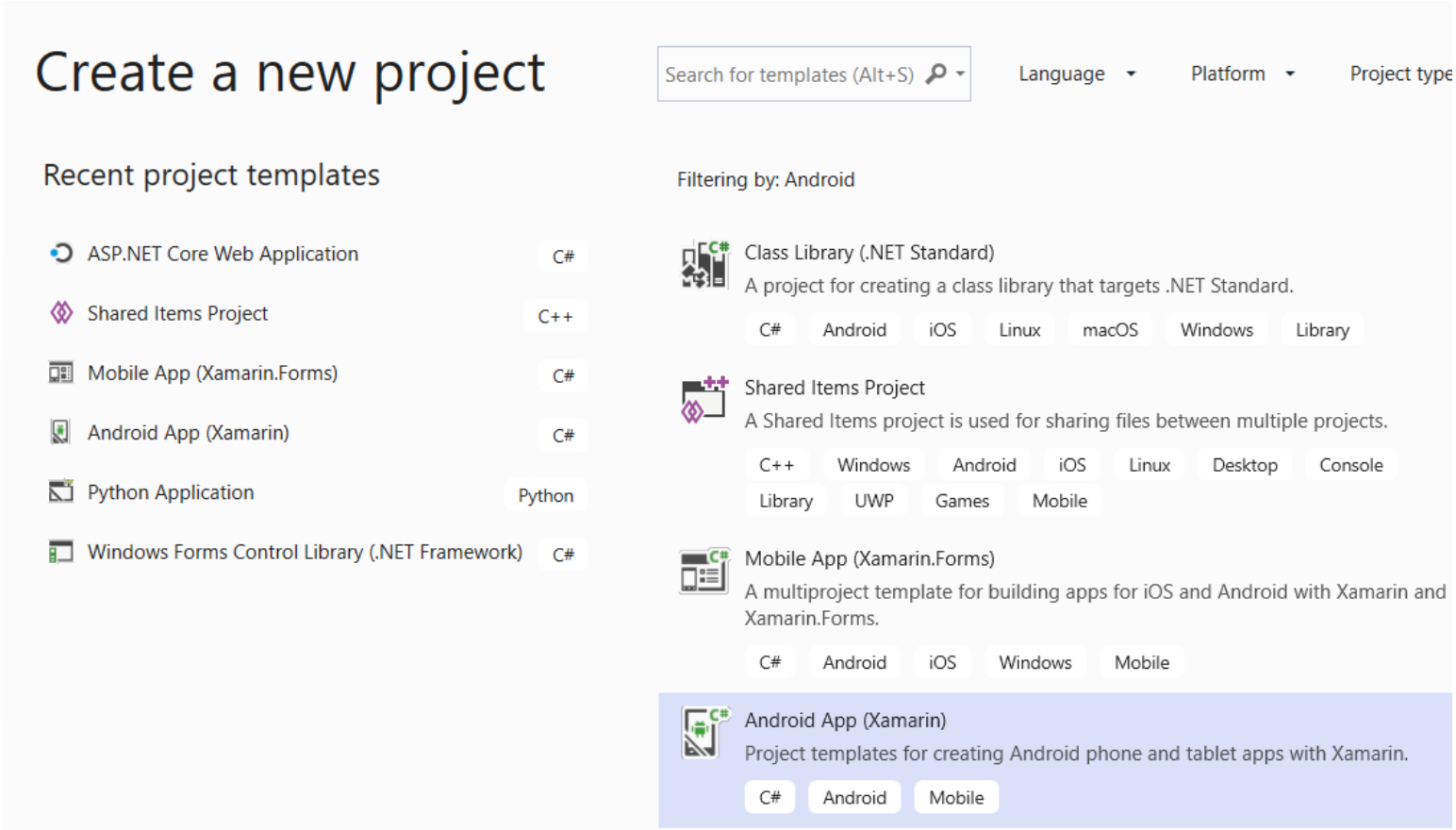
Receiving SMS



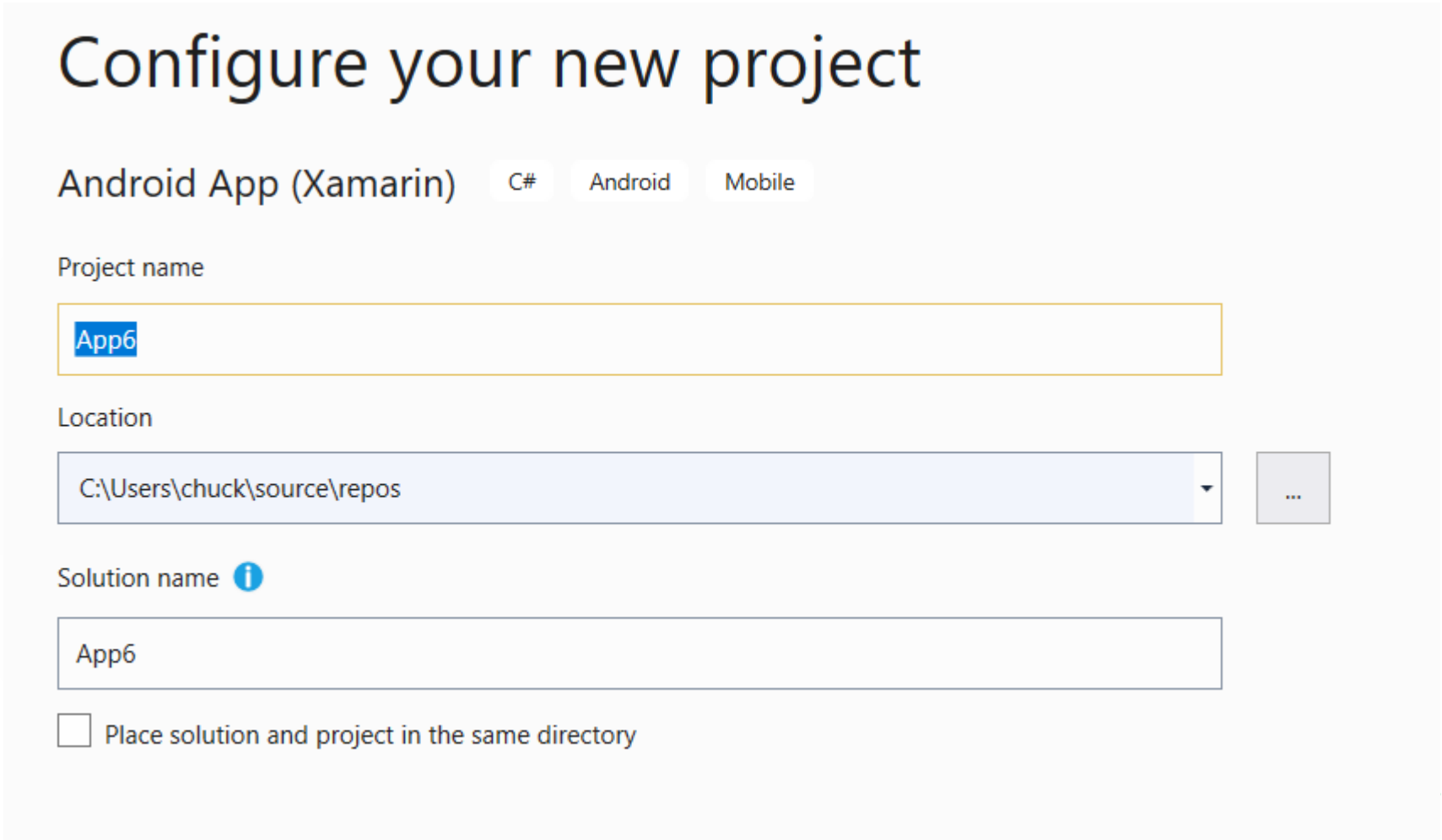
Visual Studio Android



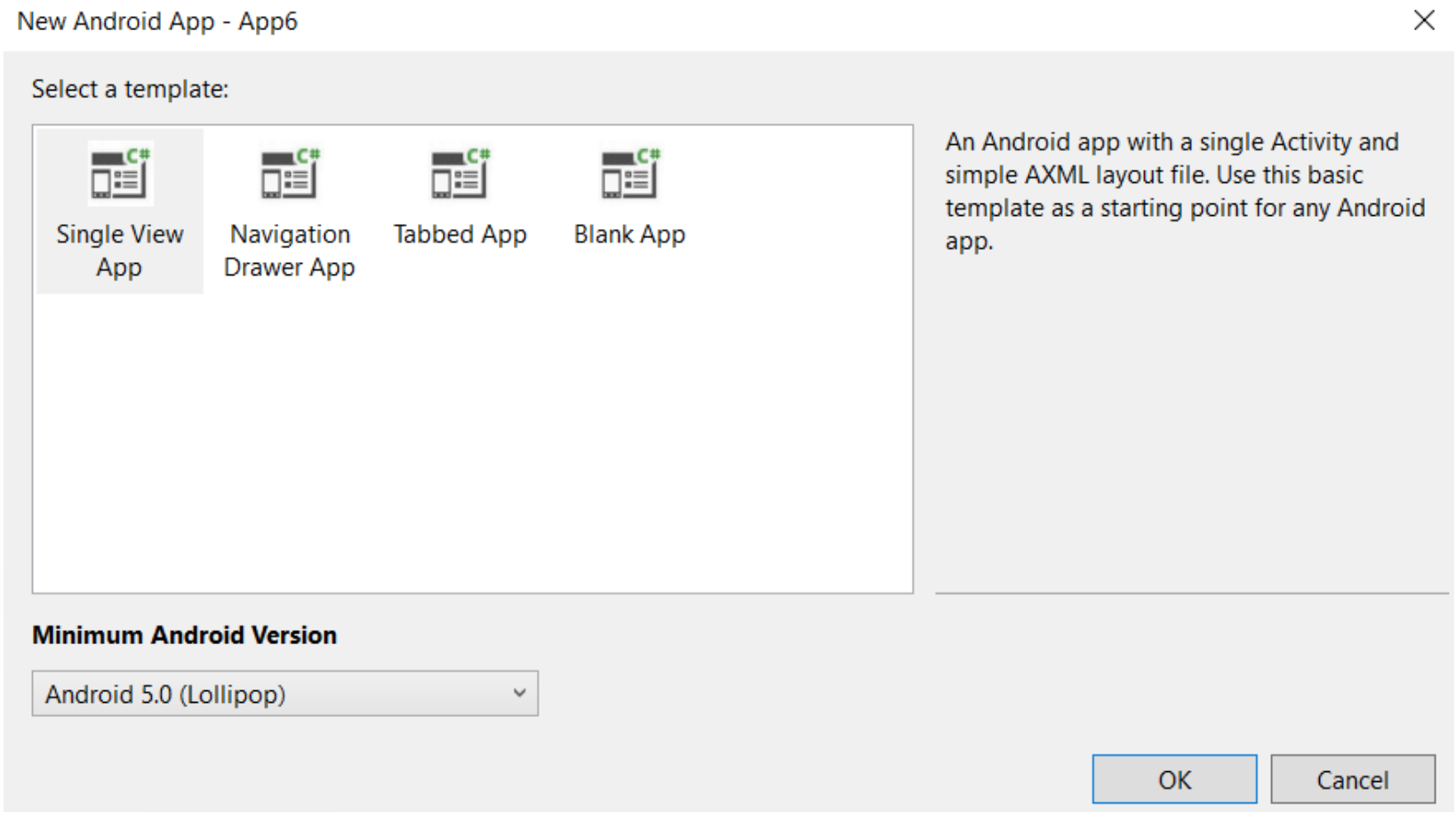
Visual Studio Android



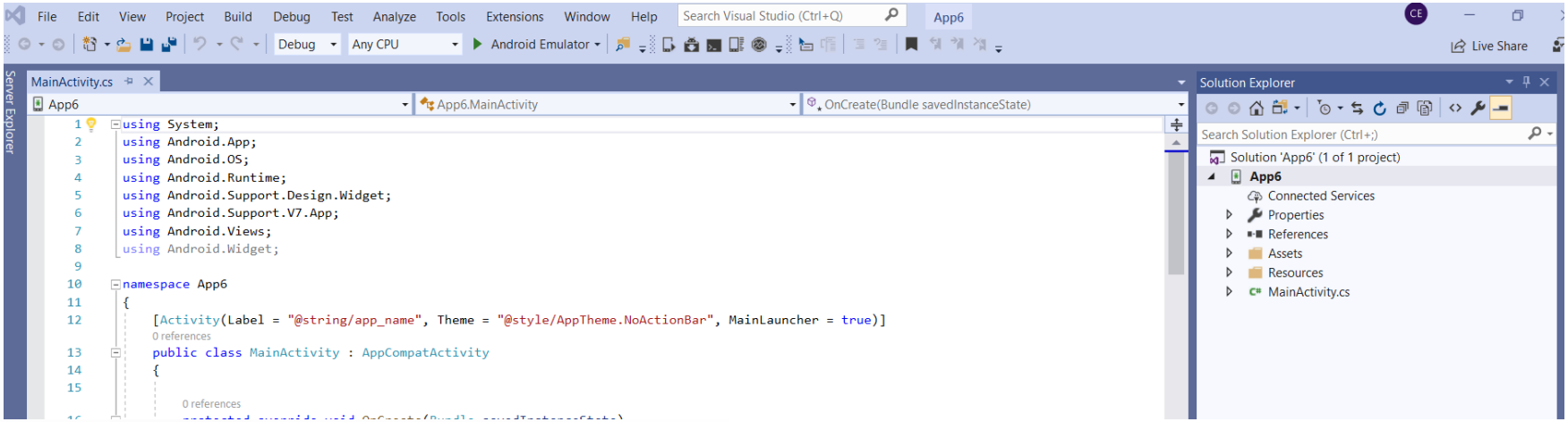
Visual Studio Android



Visual Studio Android

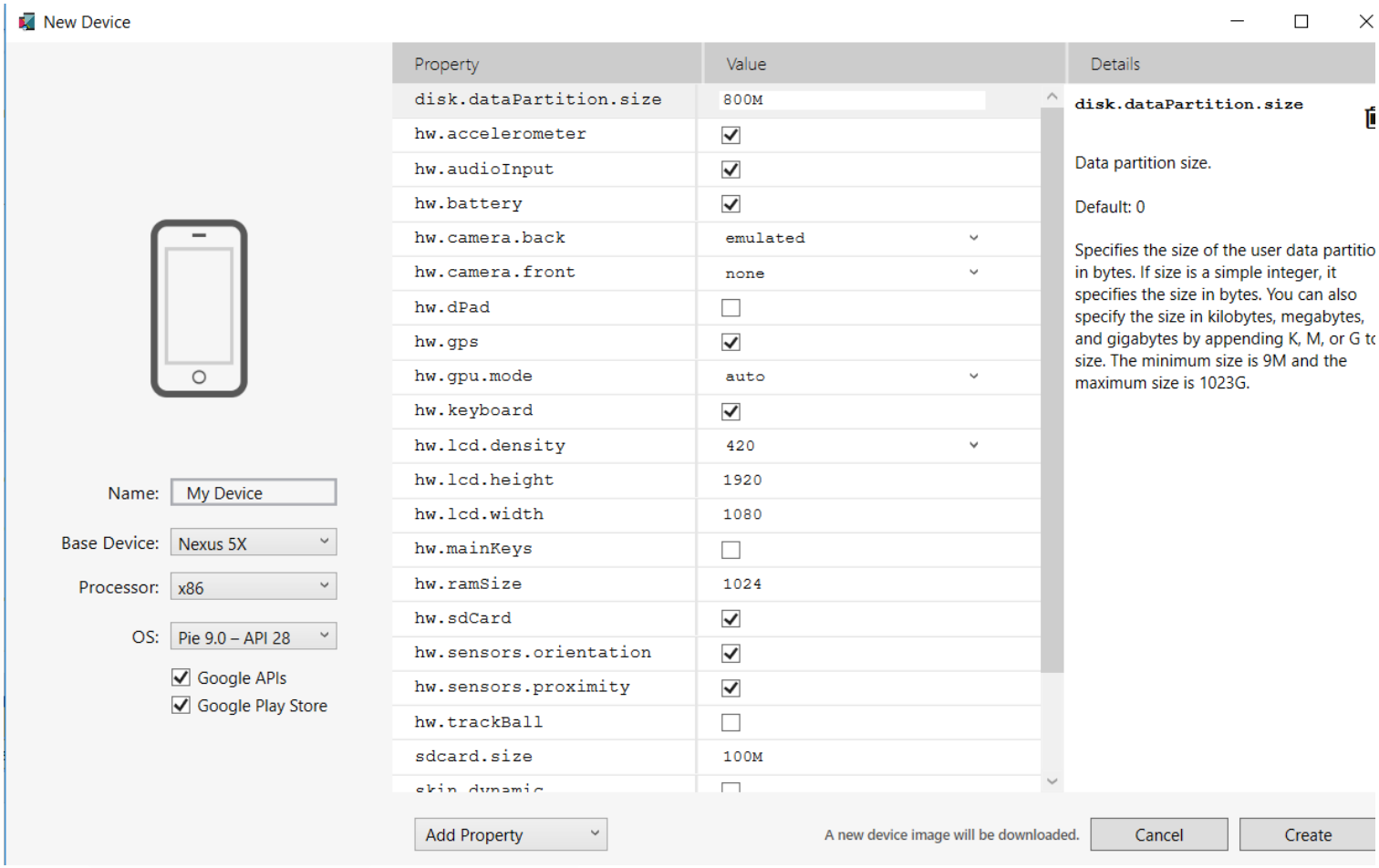


Visual Studio Android



Android

Visual Studio Android



Android

Running the Android Emulator

- The first time you run any emulator, it has to download and this may take a few minutes.