



ANDROID PROGRAMMING

App development essentials with Java programming language

COURSE CONTENTS

Overview

- Introduction to Android platform. Tools, IDE, architecture. Android Studio.
- Android versions, API levels
- Hello Android!

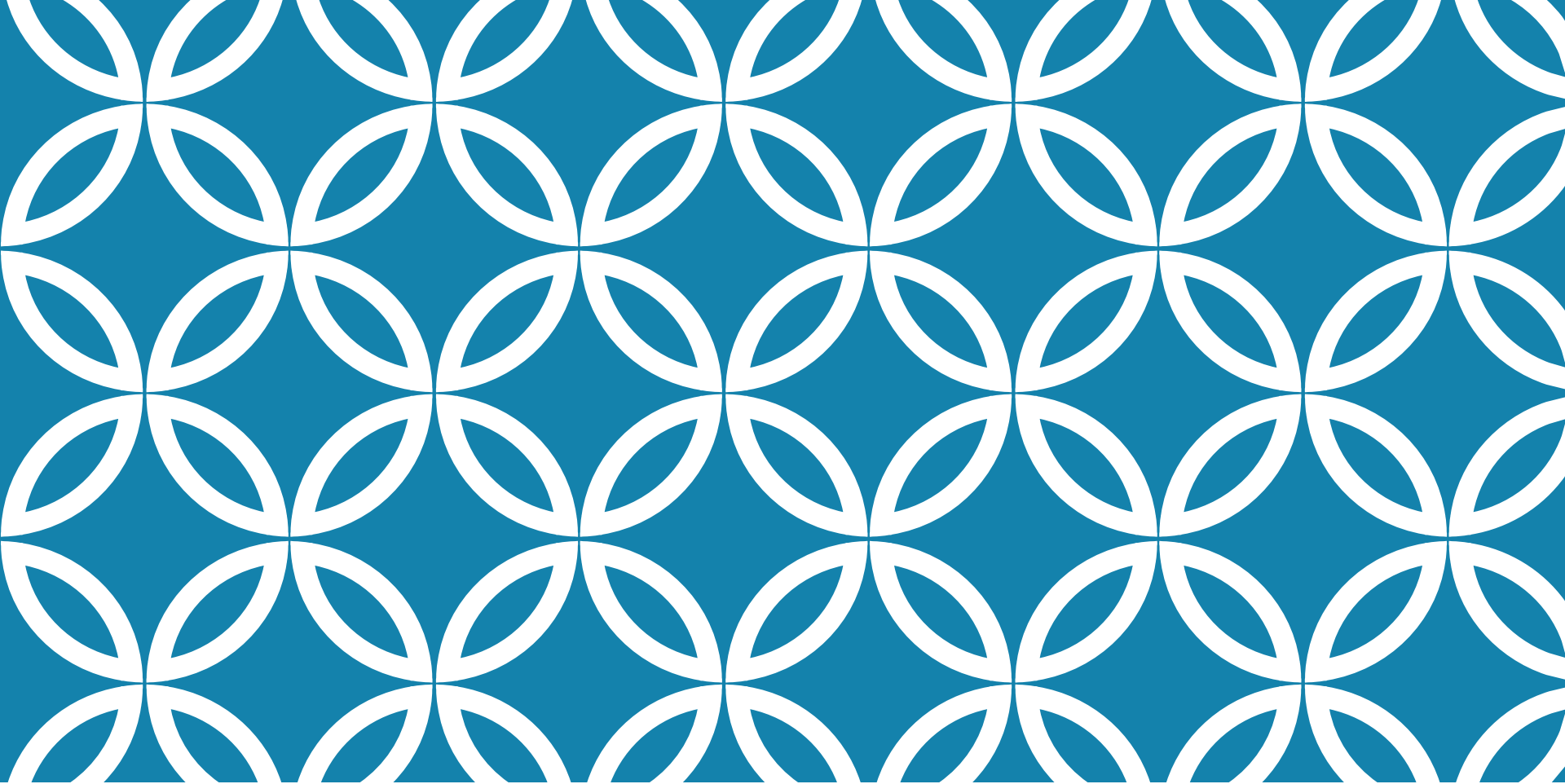
Android app development

- Android designer, UI views, layouts, event handling
- Activities, activity lifecycle
- Intents and intent filters
- Resources and localization
- Networking (consuming Rest/JSON)
- Permissions and manifest
- Utilizing device APIs (sensors and location as examples)

JAVA

Java essentials are utilized through the course

- Java language essentials: compiling, conventions and keywords, variables, control flow, optionals, functionals and lambdas.
- Classes and objects, inheritance
- Generics, collections and Exception handling



OVERVIEW



ANDROID

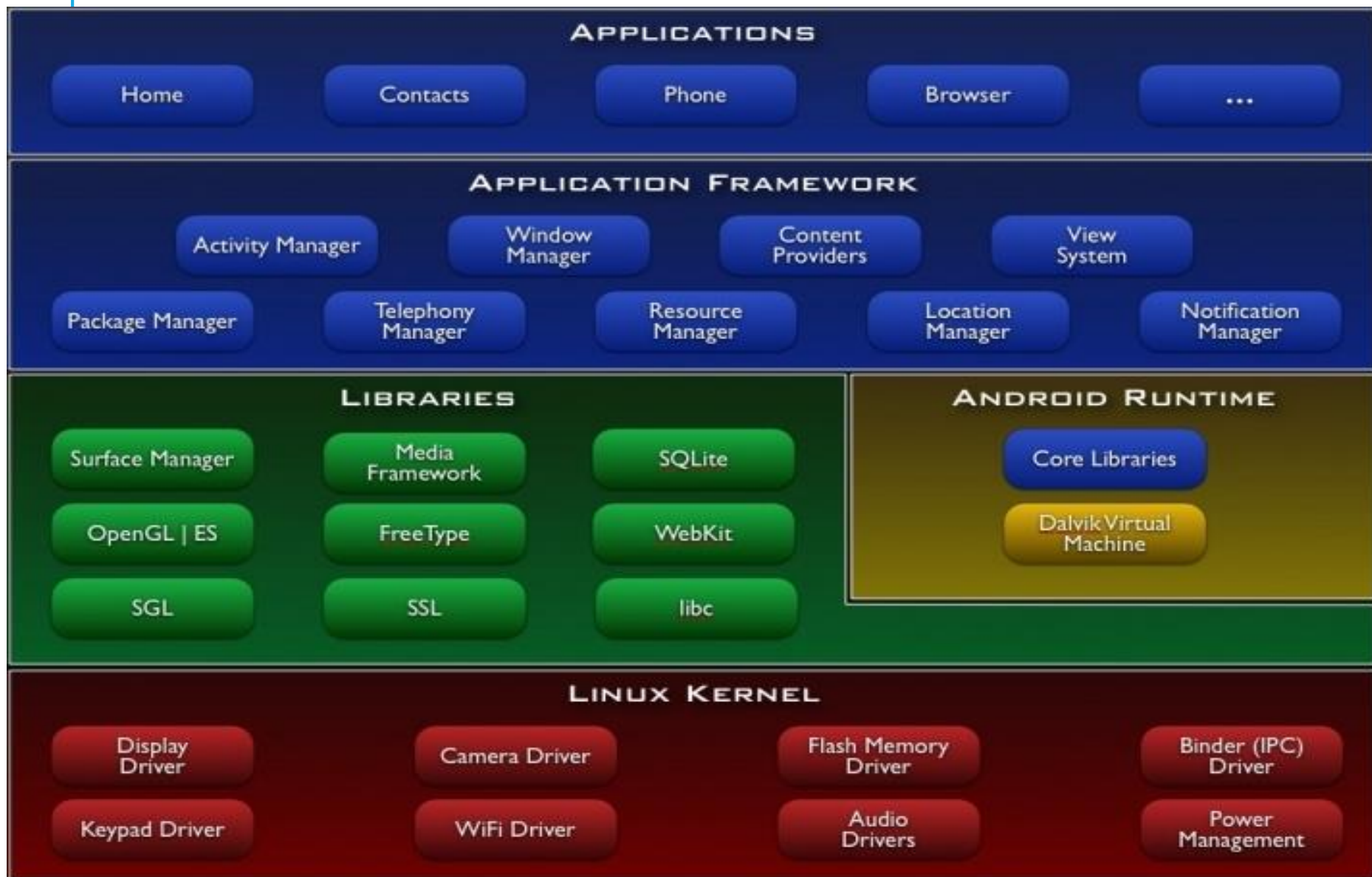
A complete set of software for mobile devices

- an operating system
- middleware
- key mobile applications

Open

All applications are equal

Boundaries between applications are low



ANDROID RUNTIME

Dalvik: Android's custom clean-room implementation virtual machine

- Designed for embedded environment

Core APIs for Java language provide a powerful, yet simple and familiar development platform

- Data structures
- Utilities
- File access
- Network Access
- Graphics
- ...

SOME ANDROID DEVICES



Huawei P10



LG G6



Samsung S8



Sony SmartWatch 3

APPLICATION BUILDING BLOCKS

Activity

- UI component typically corresponding to one screen.

Intent

- Message that triggers Activity, Service, or BroadcastReceiver.
- Used to communicate between application parts.

Service

- Faceless task that runs in the background.

BroadcastReceiver

- Set and respond to notifications or status changes.
- Can wake up an application

ContentProvider

- Enables applications to share some data

APPLICATION MODEL

In Android, there's NO strong correlation between application image and the process

- Much more fluid borders

Application, as user might see it, is effectively a **task**

Task is a collection of **activities**

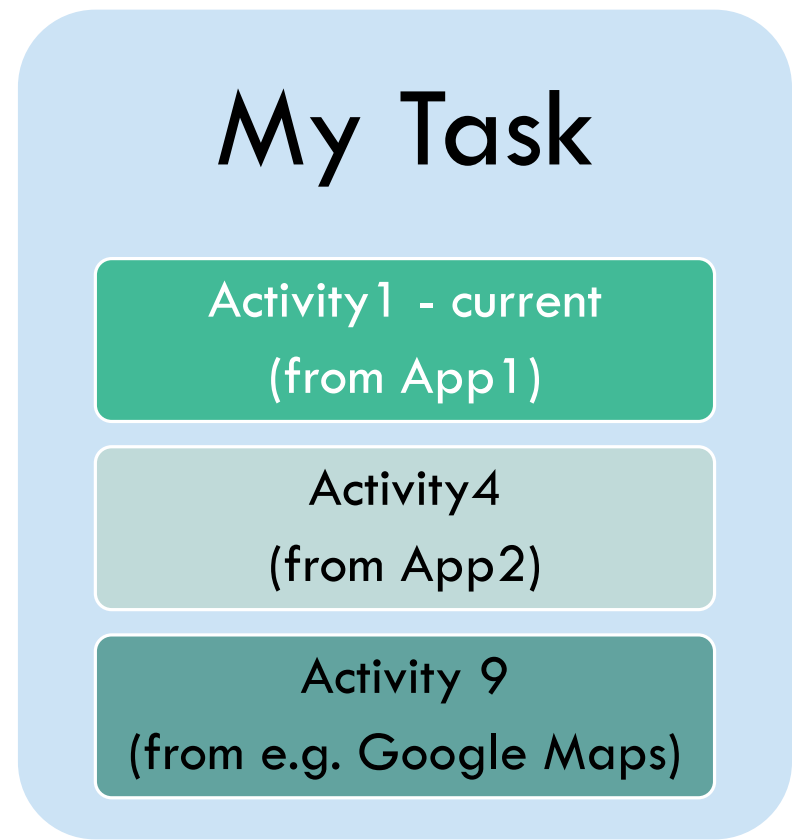
- Those that have been used by the user after the task was created and not yet closed
- Or those existing in an activity stack

ACTIVITY AND TASK

Applications and Activities



One task as Activity Stack



TASK AFFINITIES

Task Affinity is a unique static name for the task that one or more activities are intended to run in

- The default task affinity for an activity is the name of the .apk package name the activity is implemented in

To identify to whom an activity belongs to even when launching the activity to another task

- E.g. your application wants one of its' activities to replace a system activity

To separate multiple activities from which user can choose when launching the application from the .apk

PROCESSES AND THREADS

When the first of an application's components needs to be run, Android starts a Linux process for it with a single thread of execution

- By default, all components of the application run in that process and thread.
- components can be arranged to run in other processes
- additional threads can be spawned for any process

The process where a component runs is controlled by the manifest file

Any process may be shut down by Android at some point

- E.g. when memory is low and required by other processes that are more immediately serving the user

Since everything, including UI, runs in the main thread, avoid long lasting operations there

- Maintain UI responsive
- Launch another thread for the long-lasting operation
- Multi-threading and Remote Procedure Calls will be discussed in more detail later

SERVICE

Service is an application (or its component) running in the background without user interaction

Service run on application's thread

- Should launch a dedicated thread for e.g. CPU intensive services

To start a new service owned by this context, use

```
Context.startService()
```

- To connect to another service or start it if it's not running, use:

```
Context.bindService()
```

CONTENT PROVIDER

Mechanism to share data between applications

- Only way to share data between packages

Simply a class with a set of standard methods

- all content providers must implement a common convention to query for data, and a common convention to return results
- Internal structure is up to content provider

common data types supported by the platform

- audio, video, images, personal contact information etc.
- See package **android.provider**

BROADCAST RECEIVER

Intent receiver that will react to system-wide messages and events

Not visible, but works on the background like Services

Currently requires application to be started to react to events

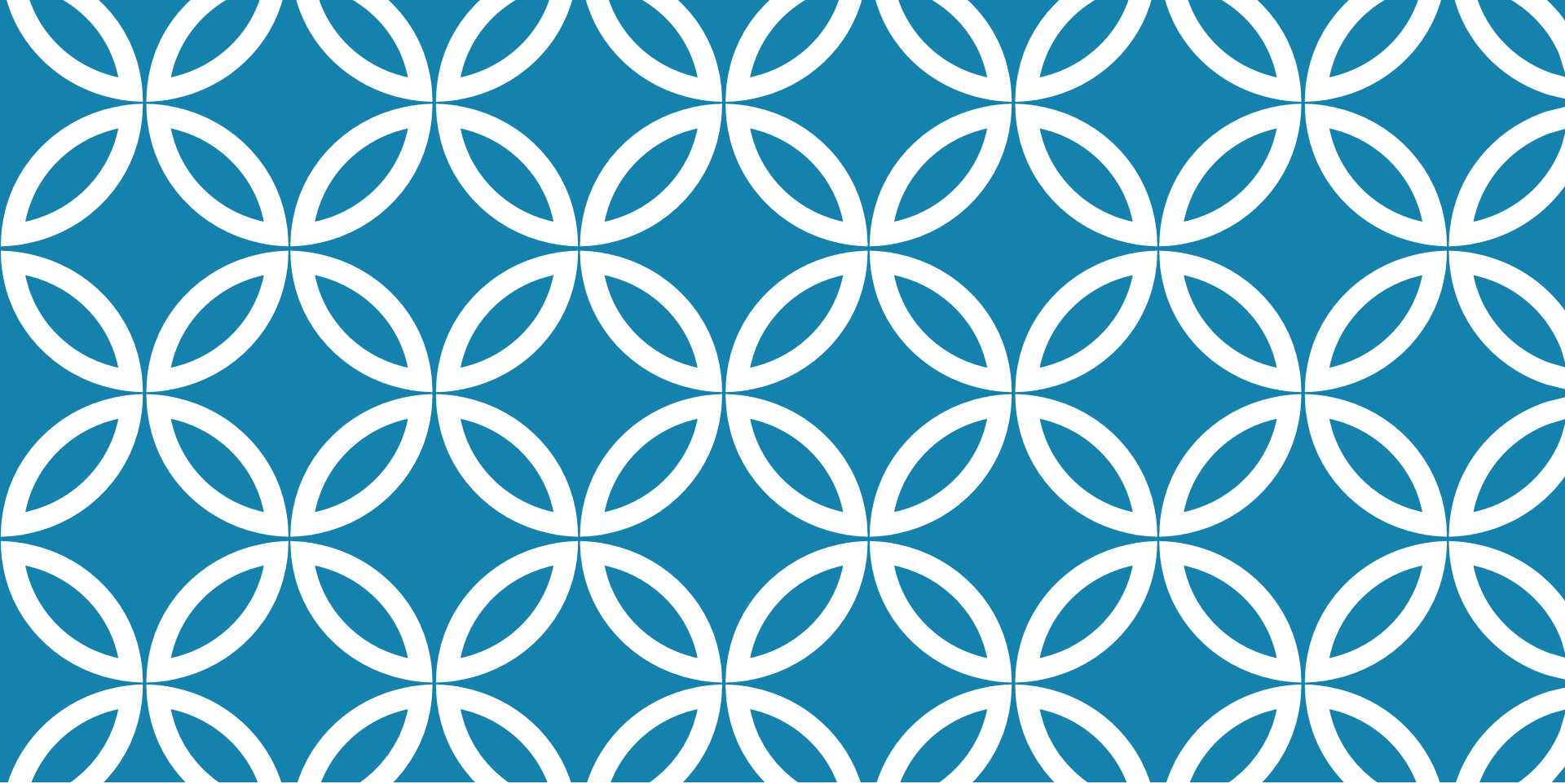
Events can be incoming SMS messages, changes in location or status of available system services, etc.

SUMMARY

Android is the first complete, open, and free mobile platform provided by the Open Handset Alliance

Android is built on the Linux kernel, but Android is not Linux

Application Framework exposes the same APIs to both 3rd party apps and core components



GETTING STARTED WITH DEVELOPMENT



ANDROID STUDIO

The official IDE for Android

For all device types: Phones, Tablets, Wearables...

Supported platforms

- Windows (64 & 32 bit)
- MAC
- Linux

Download and installation instructions from:

<https://developer.android.com/studio/index.html>

SOME ANDROID STUDIO FEATURES

Code editor – with intellisense

UI Design tools and layout editor

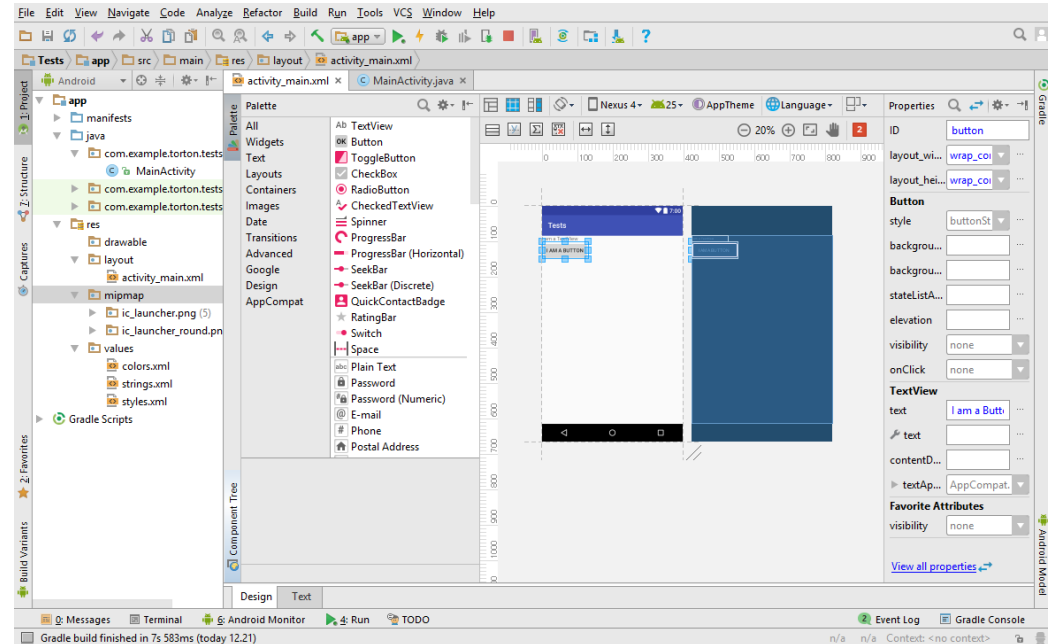
Debugger

Testing tools

Profiler

Instant Run

Emulator



INSTALLING ANDROID STUDIO

Installation instructions can be found from:

<https://developer.android.com/studio/install.html>

DEPLOYING AND TESTING APPS

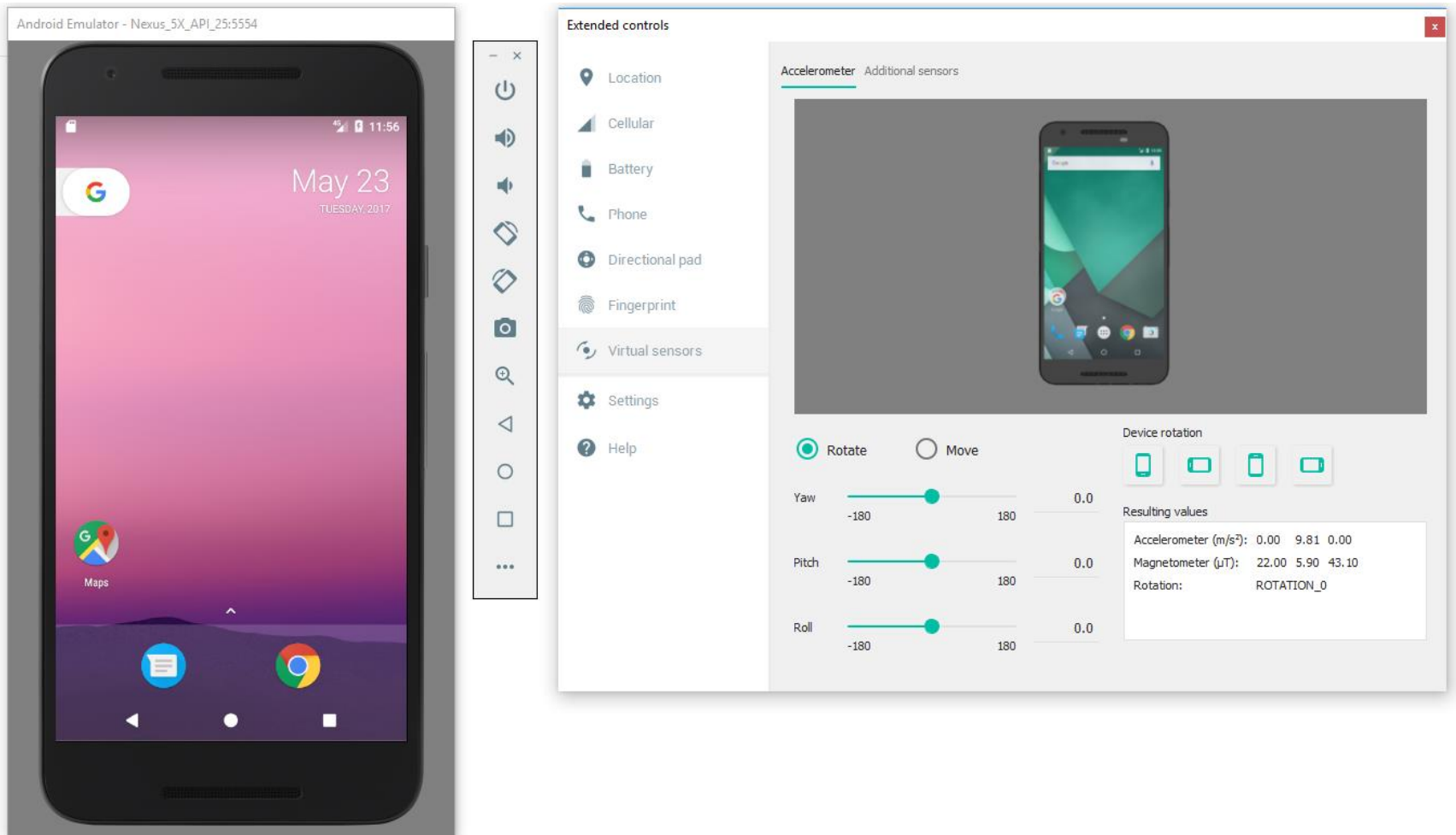
Apps can be run and debugged within Android Studio by using

- Android Emulator
- Hardware device via USB (phone, tablet etc.)

If you use a HW device e.g. your phone for debugging, you need to set the phone into "developer mode"

- Can be done from device settings
- Check your phone vendor for detailed instructions

ANDROID EMULATOR



EMULATING HW FEATURES

The emulator contains many virtualised hardware features for testing including

- Location
- HW Sensors
- Battery
- Networking
- Phone & Messaging

Some hardware features can only be tested on a real device

- WiFi
- Bluetooth
- NFC
- SD Card
- Headphones
- USB

CREATING AN ANDROID PROJECT

- Select File -> New -> Android Project

Name of the application

Package namespace

Directory or folder to contain the project.

Create New Project

New Project

Android Studio

Configure your new project

Application name: My Application

Company domain: mydomain.example

Package name: example.mydomain.myapplication [Edit](#)

☐ Include C++ support

Project location: C:\AndroidProjects\HelloWorld

Previous Next Cancel Finish

SELECT TARGET DEVICES

- Select the device types and minimum SDK API levels for your project

Specifies the API level that is required on device to install application to it.



Create New Project

Target Android Devices

Select the form factors your app will run on

Different platforms may require separate SDKs

☒ Phone and Tablet

Minimum SDK: API 21: Android 5.0 (Lollipop)

Lower API levels target more devices, but have fewer features available. By targeting API 21 and later, your app will run on approximately 40.5% of the devices that are active on the Google Play Store. [Help me choose](#)

☐ Wear

Minimum SDK: API 21: Android 5.0 (Lollipop)

☐ TV

Minimum SDK: API 21: Android 5.0 (Lollipop)

☐ Android Auto

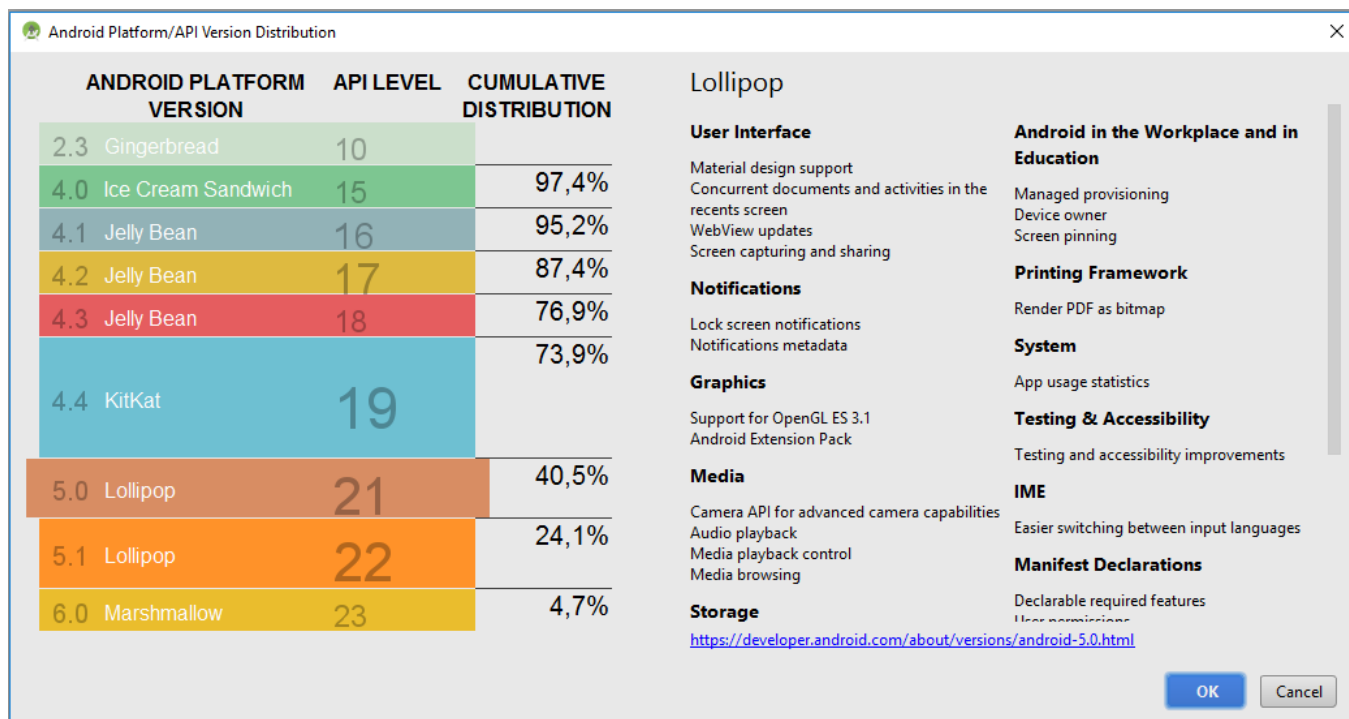
Previous Next Cancel Finish

SELECTING MINIMUM API LEVEL

Select the minimum API level depending on the features and APIs you use

The smaller the level, the more devices you can cover

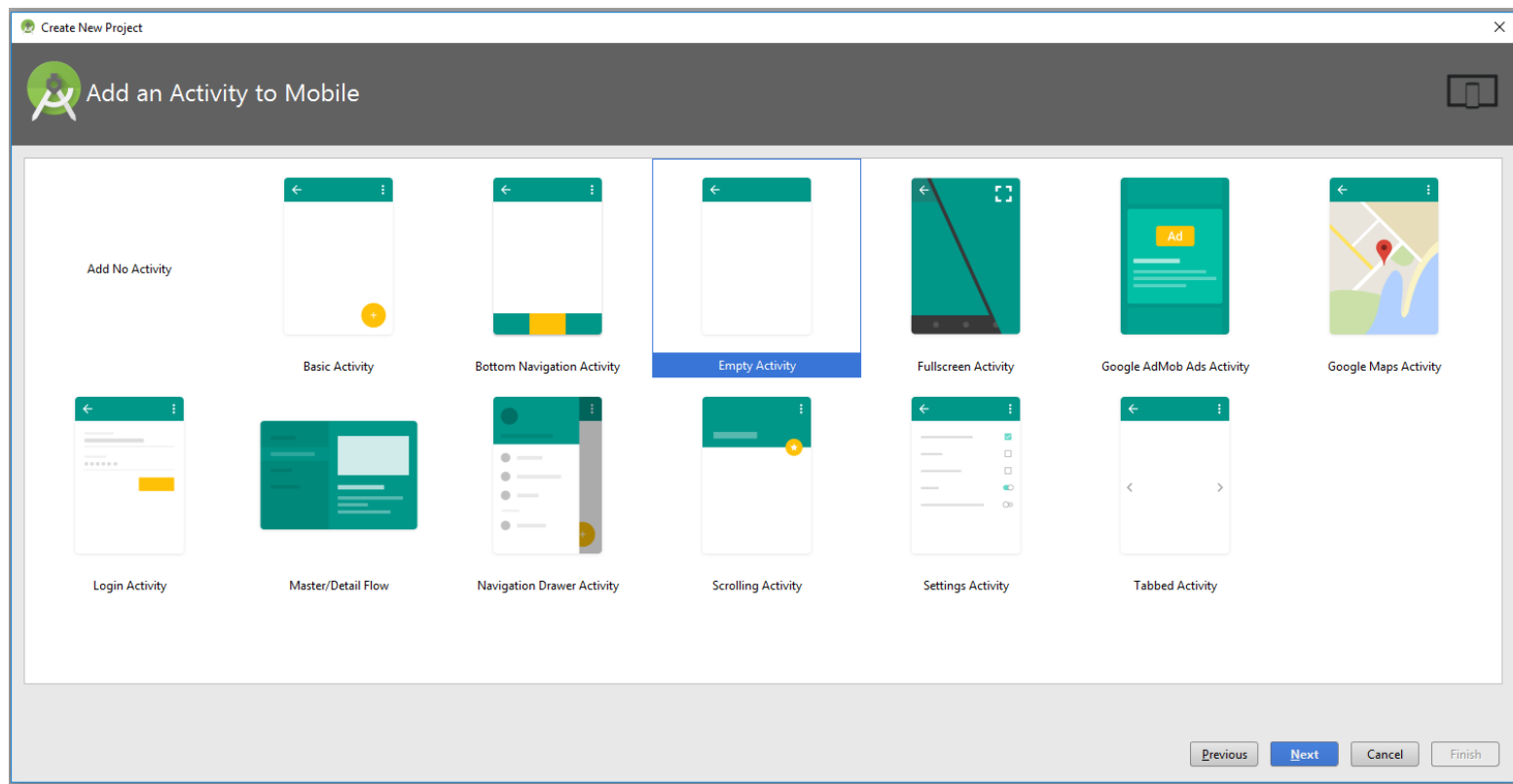
E.g. 73.9% of devices are capable of running API level 19 (KitKat)



ADDING THE FIRST ACTIVITY

There are number of pre-written Activity templates you can start with

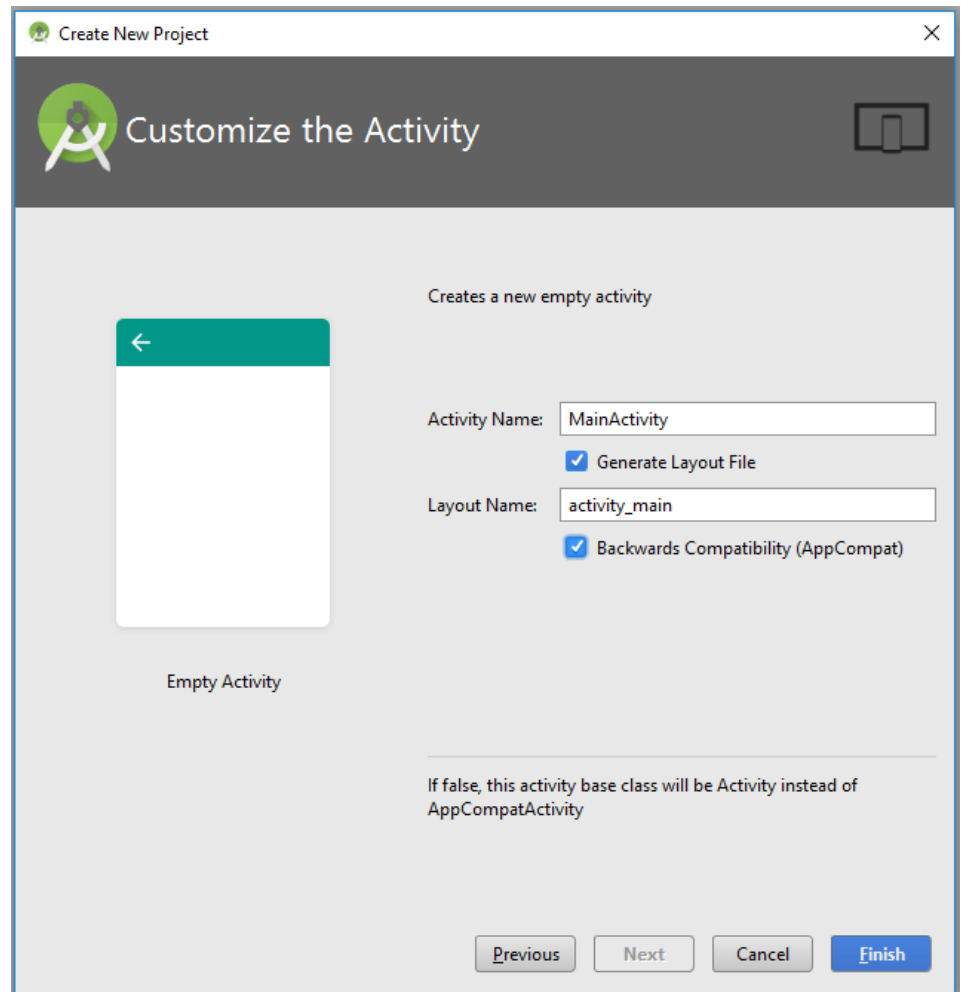
We can use Empty Activity for the first HelloWorld



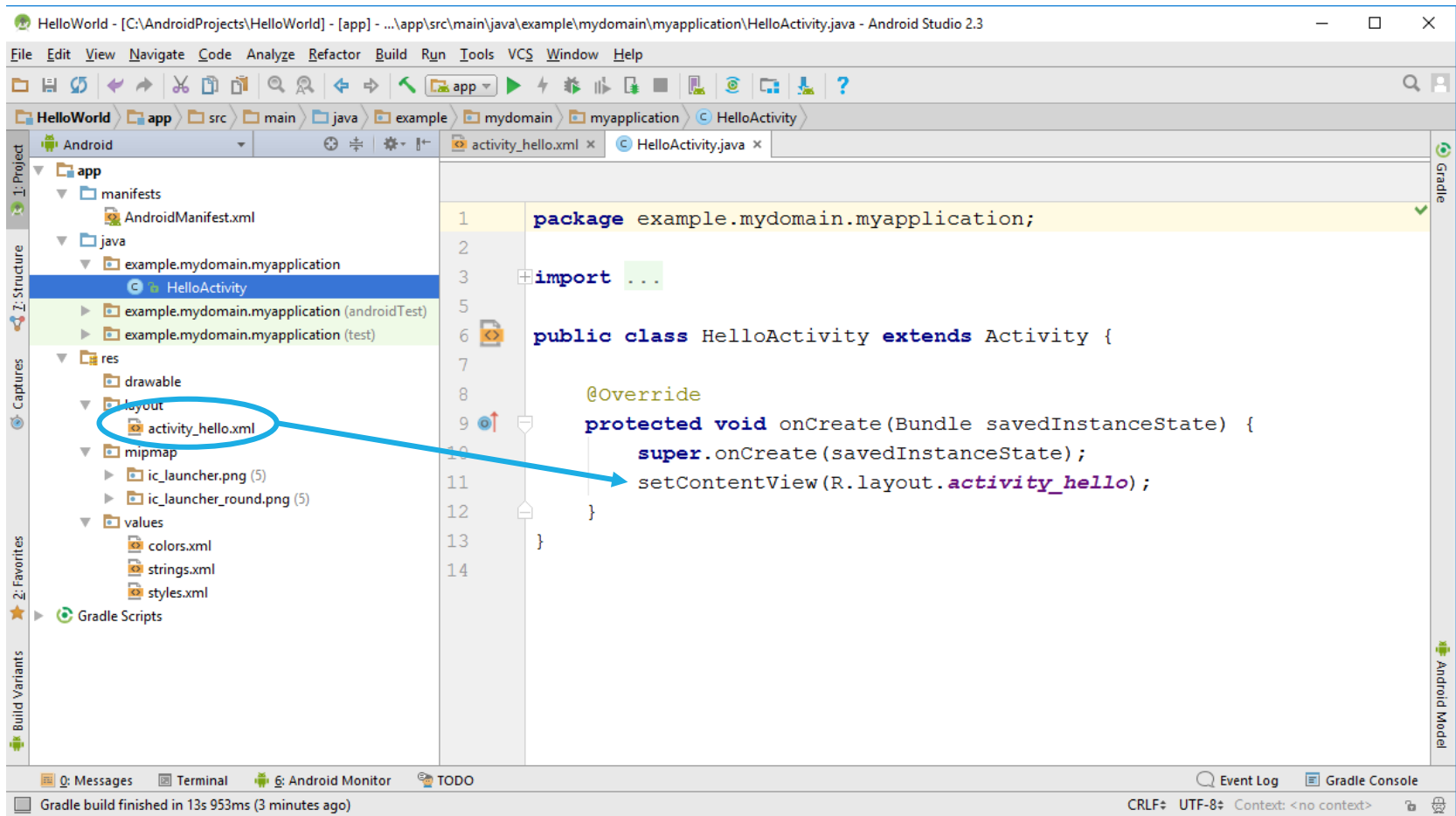
ADDING THE FIRST ACTIVITY

"Generate Layout File" generates an XML file for your Activity UI

With XML layout file, you can use Android Designer tool and XML to develop the UI instead of Java or Kotlin

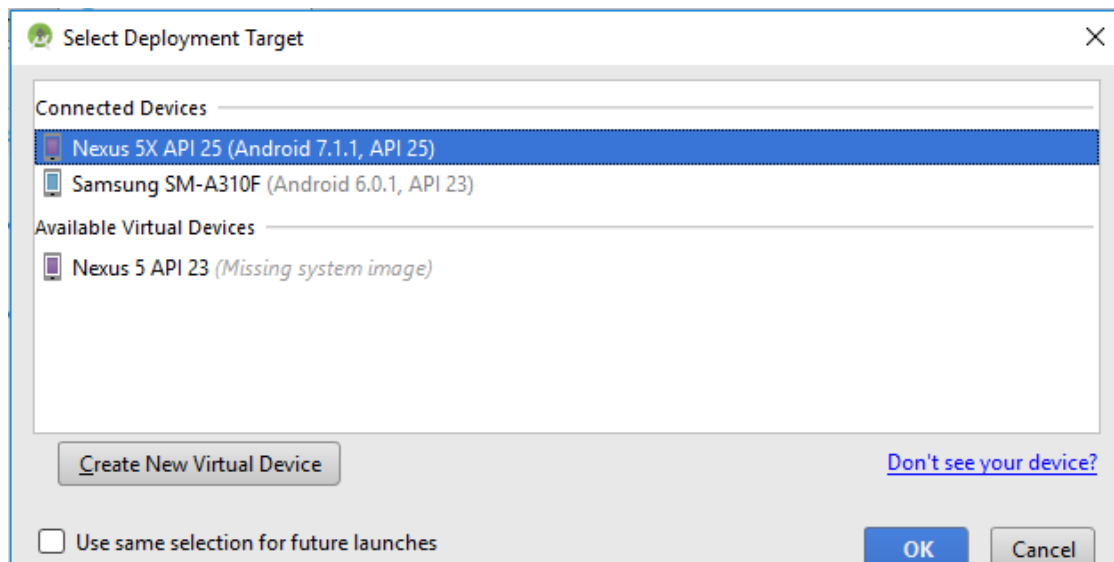


THE STRUCTURE OF ANDROID PROJECT



RUNNING (AND BUILDING)

- Select Run -> Run 'App'
- The next step is to select the target Android device to install and run the application
- Android studio detects all installed Emulator Virtual Machines and real devices (in developer mode) connected with USB
- New virtual machines can be added (and needs to be added in the first time) if the app is ran in an emulator




ANDROID EMULATOR AND AVD MANAGER
















Apps can be tested on you PC without a real device



Runs a full Android system stack (same system image as in your device)

Use same tool chain to work with devices and emulator

You can create different emulator hardware profiles via AVD (Android Virtual Device) configuration tool

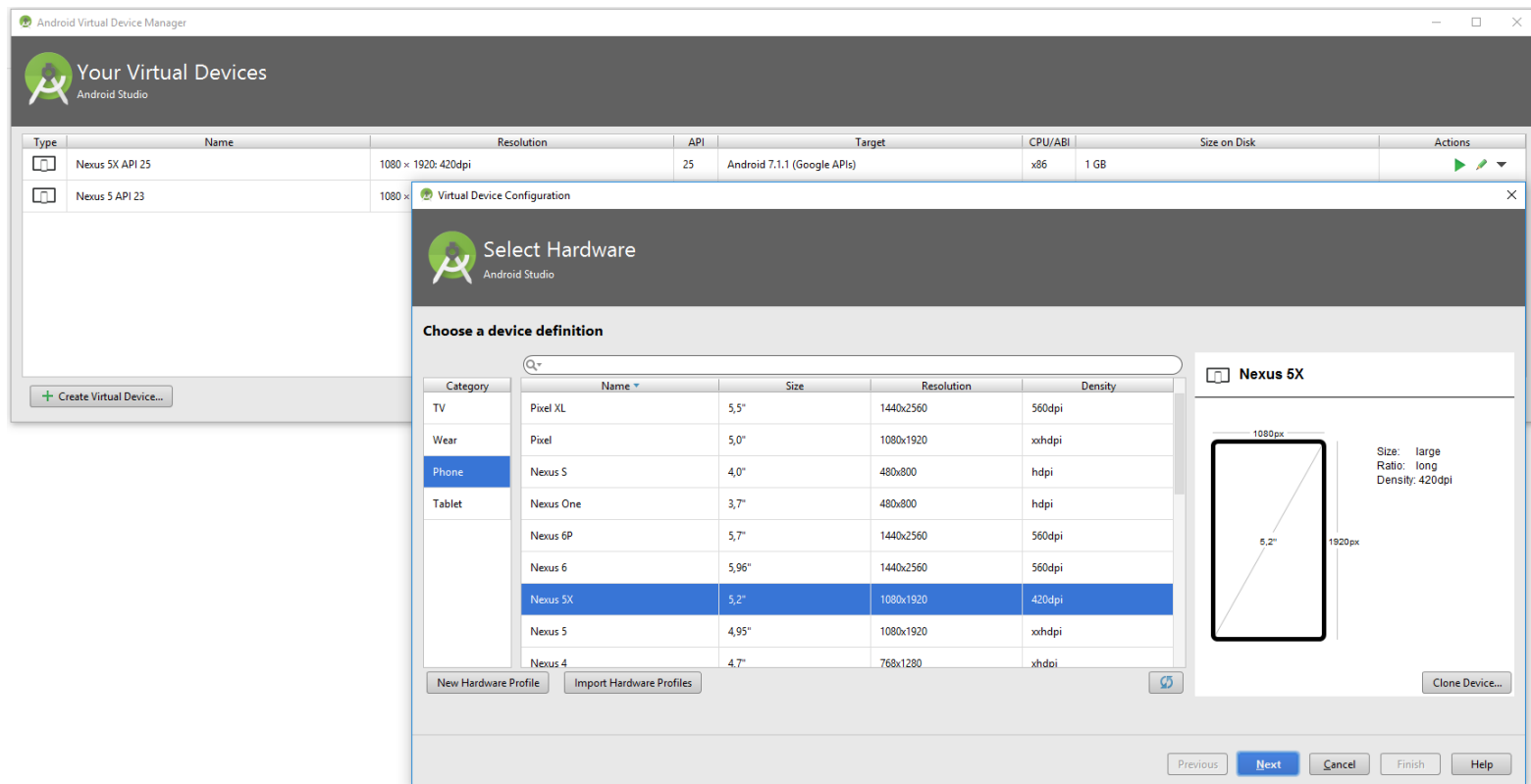
 **Your Virtual Devices**
Android Studio

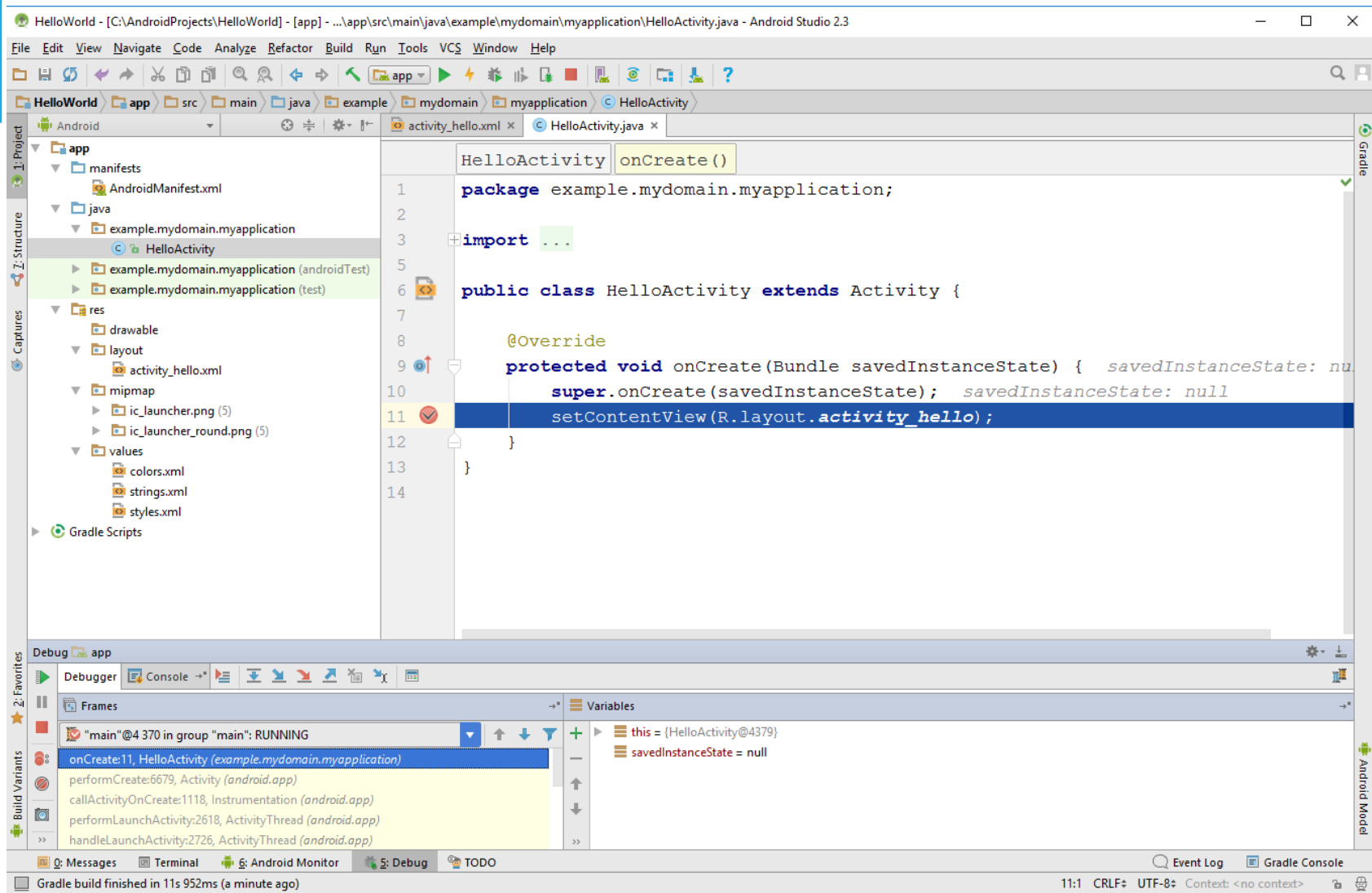
Type	Name	Resolution	API	Target ▲	CPU/ABI	Size on Disk	Actions
	Android Wear Square API 21	280 × 280: hdpi	21	Android 5.0 (Android Wear)	x86	1 GB	  ▼
	Nexus 5 API 22	1080 × 1920: xxhdpi	22	Android 5.1 (Google APIs)	x86	1 GB	  ▼
	Android TV (1080p) API 23	1920 × 1080: xhdpi	23	Android 6.0 (Android TV)	x86	1 GB	  ▼
	Nexus 5X API 23	1080 × 1920: 420dpi	23	Android 6.0 (Google APIs)	x86	1 GB	  ▼
	Nexus 9 API 23	2048 × 1536: xhdpi	23	Android 6.0 (Google APIs)	x86	1 GB	  ▼

 + Create Virtual Device... 

ADDING NEW VIRTUAL DEVICES

Create an Android Virtual Device using Android SDK and AVD Manager (Tools->Android->AVD Manager)





UI: JAVA/KOTLIN (CODING) OR XML (DESIGNER) BASED?

Creating UIs programmatically in code is the traditional way

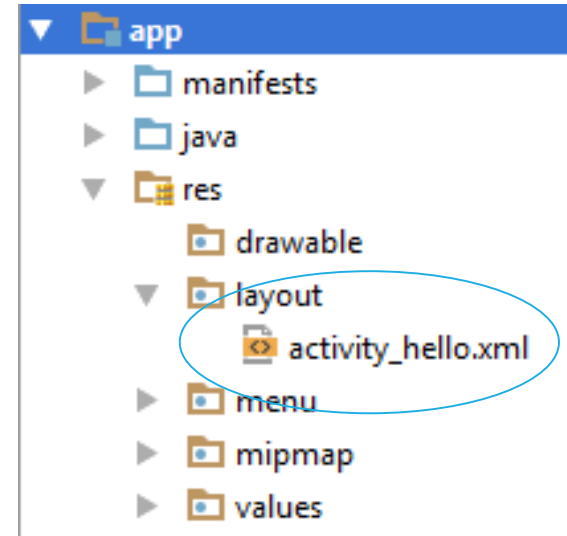
- Complex UIs hard to layout in code
- Can be challenging to maintain

The primary option is to use XML based layout files

- UI structure defined in XML file
- Behaviour in code
- Easy to localize
- More platform independency
- Quick (Android Designer)

HELLO WORLD! — XML LAYOUT

Create an XML file specifying the layout



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

    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello World!"
/>
```

FINDING RESOURCES IN CODE

- A project's R.java file is an index into all the resources defined in the file

```
public final class R {  
    public static final class attr {  
    }  
    public static final class drawable {  
        public static final int icon=0x7f020000;  
    }  
    public static final class layout {  
        public static final int main=0x7f030000;  
    }  
    public static final class string {  
        public static final int app_name=0x7f040001;  
        public static final int hello=0x7f040000;  
    }  
}
```

SUPPORT LIBRARIES

If you want your code to work both in old and new devices, and use as much new features as possible, you can include Support Library in your project

To use, you must first download the support packages with Android SDK Manager

- And then add them to your project in IDE or your build manager

Your application is now ready to use the library APIs. All the provided APIs are available in the `android.support` package (for example, `android.support.v4`)

MOST IMPORTANT SUPPORT LIBRARY CLASSES

ConstraintLayout

AppCompatActivity

Fragment

FragmentManager

FragmentTransaction

ListFragment

DialogFragment

LoaderManager

Loader

AsyncTaskLoader

CursorLoader

GRADLE BUILD SYSTEM

Android Studio comes with Gradle build system

Gradle features can be used to

- Customize, configure, and extend the build process
- Create multiple APKs for your app, with different features using the same project and modules
- Reuse code and resources across sourcesets

For more info, visit

<https://developer.android.com/studio/releases/gradle-plugin.html>

SUMMARY

SDK installation and upgrade

Development environment

Hello Android! Application

Running on emulator [and on device]

Using the debugger

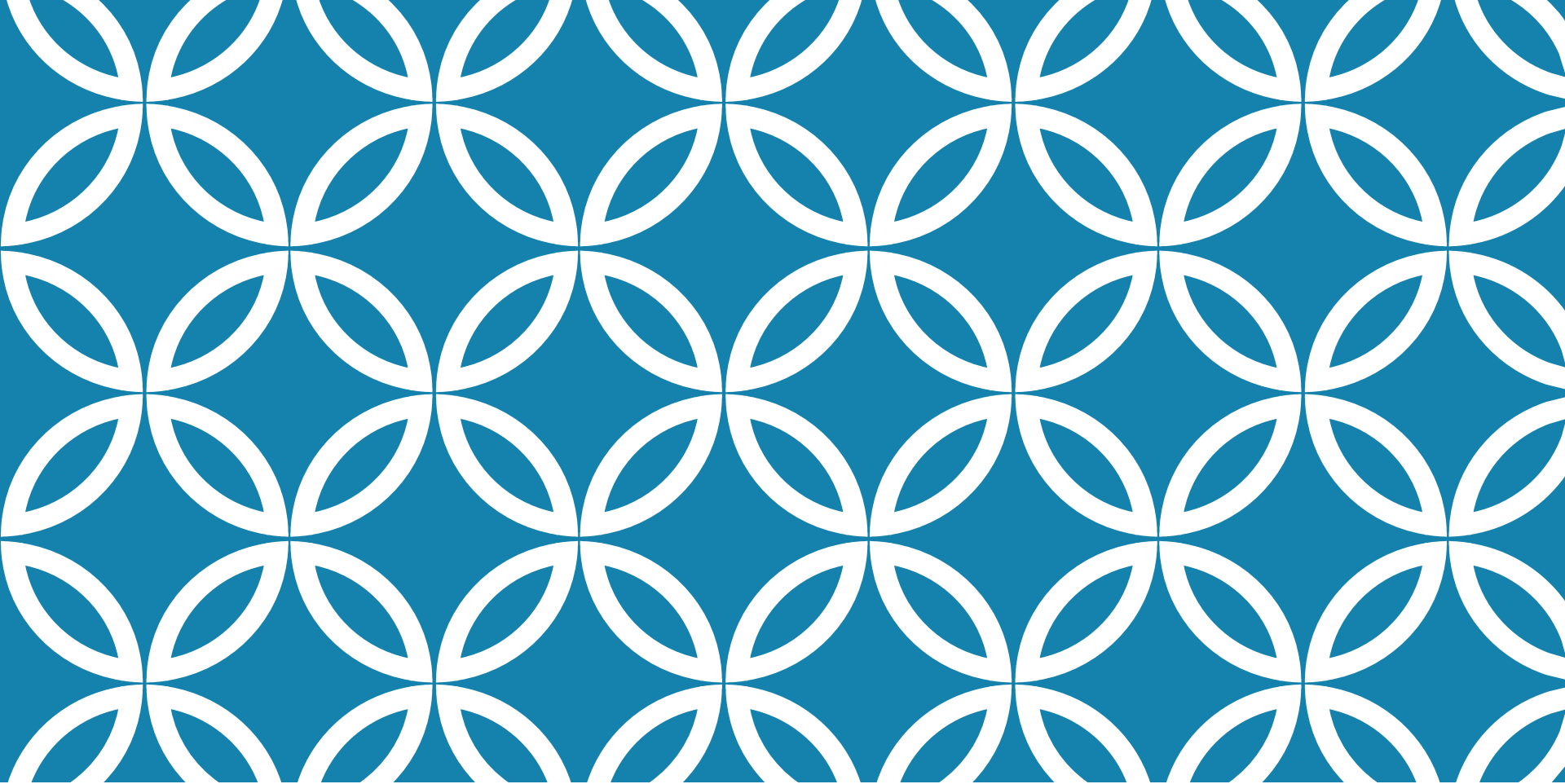
LAB: FIRST ANDROID APP

Create a new HelloWorld project for Android

Verify that everything works

Go through your environment, familiarize yourself with

- Project structure and file locations
- Emulator controls
- Android Studio features



APPLICATION UI ESSENTIALS

Android Essentials



MODULE CONTENTS

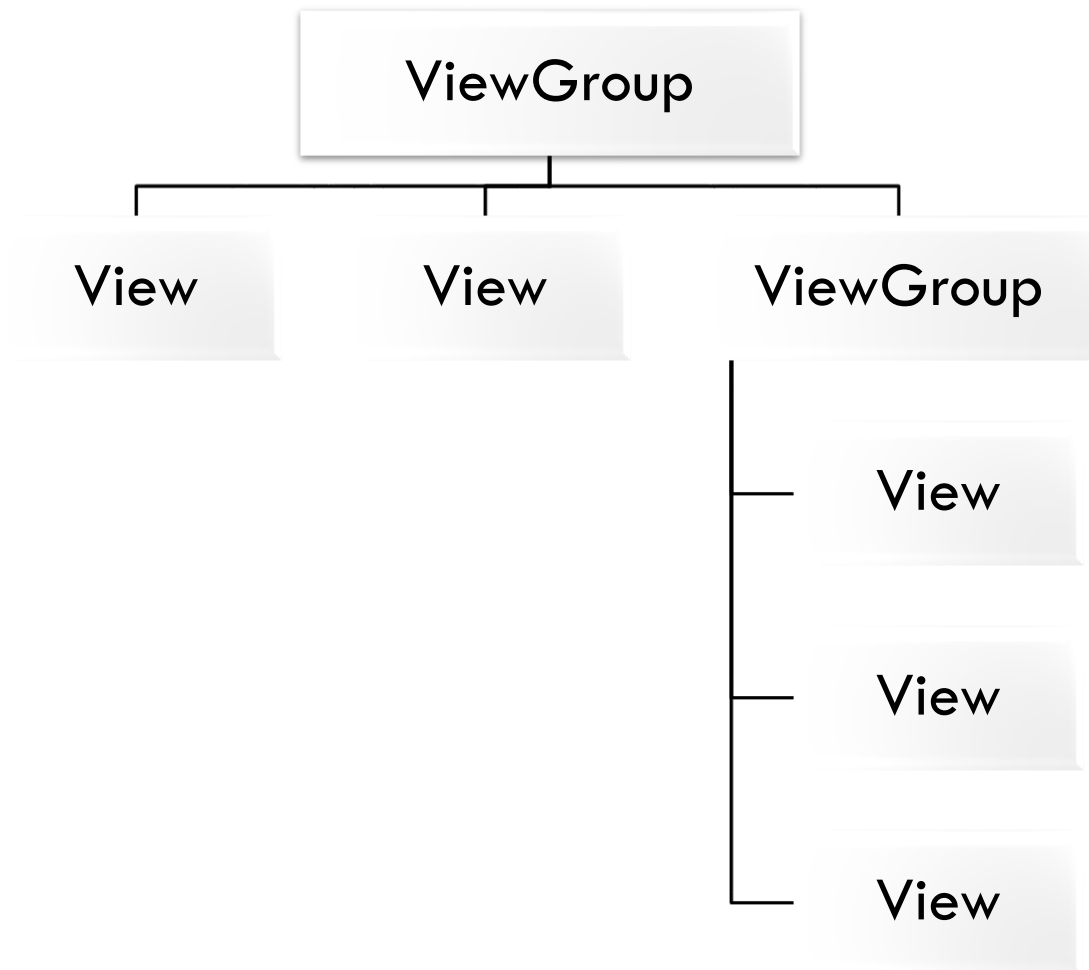
Ui Views

Layouts

Event handling

Resources and localization

VIEWS AND VIEW GROUPS



DECLARING THE LAYOUT

Two different ways available:

- **Declare UI elements in XML.**
- **Instantiate layout elements at runtime.**

XML provides full isolation between layout and behaviour

- One approach is to define basic layout on XML and state dependent layout changes are implemented in code

ADT provides a simple editor for the purpose

EXAMPLE LAYOUT XML FILE

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

    android:layout_width="fill_parent"

    android:layout_height="fill_parent"

    android:orientation="vertical" >

    <TextView android:id="@+id/text"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:text="I am a TextView" />

    <Button android:id="@+id/button"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:text="I am a Button" />

</LinearLayout>
```

Source file
format

Android
namespace

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
<Button android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="I am a Button" />
</LinearLayout>
```

Compiled resource
datatype

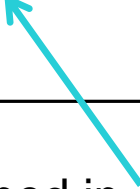
Reference to
another resource

LOAD THE XML RESOURCE

XML layout files are compiled into **View** resources

- load the layout resource from your application code
- Implement Activity.onCreate() callback, by calling setContentView()

```
fun onCreate(savedInstanceState: Bundle) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.my_activity)  
}
```



Assume that the View is defined in ***my_activity.xml***

ID OF A VIEW

// XML

<Button

android:id="@+id/my_button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/my_button_text"

/>

// Java

```
public void onCreate(Bundle savedInstanceState) {  
    ...  
    Button myButton = (Button) findViewById(R.id.my_button);  
}
```

TUNING THE LAYOUT

XML layout attributes named **layout_something** define layout parameters

- contains property types that define the size and position for each child view, as appropriate for the view group

It's possible to retrieve the position information of a View with several different methods

The size, padding and margins of a View can be set, or queried, in many different ways

SOME COMMON ANDROID VIEWS

Button

TextView

EditText

ListView

ImageView

ProgressBar

CheckBox

RadioButton

CalendarView

DatePicker

SOME EXISTING LAYOUTS

LinearLayout – for simple cases

ConstraintLayout – this is the one to use to make responsive UIs!

GridLayout

FrameLayout

DrawerLayout

RelativeLayout

TableLayout

AbsoluteLayout

UI EVENTS

Two ways to get informed about user's actions with UI components

- Define an event listener and register it with the View, or
- Override an existing callback method for the View

Using listeners:

- Register the view to receive events with **View.OnClickListener()**, **View.OnTouchListener ()** and **View.OnKeyListener()**
- Implement the callback method, e.g. **onClick()** for **OnClickListener**

Override existing callback methods

- Example events **onTouchEvent()**, **onTrackballEvent()**, **onKeyDown()**

UI EVENTS, OPTION 1: USING AN ANONYMOUS LISTENER (JAVA PATTERN)

```
// Create an anonymous implementation of OnClickListener
private OnClickListener mOnClickListener = new OnClickListener() {
    public void onClick(View v) {
        // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedInstanceState) {
    ... // Capture our button from layout
    Button button = (Button)findViewById(R.id.example);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mOnClickListener);
    ...
}
```

UI EVENTS, OPTION 2: IMPLEMENTING THE LISTENER INTERFACE

```
public class ExampleActivity extends Activity
    implements OnClickListener {
    protected void onCreate(Bundle savedInstanceState) {
        ...
        Button button = (Button)findViewById(R.id.example);
        button.setOnClickListener(this);
    }
    // Implement the OnClickListener callback
    public void onClick(View v) {
        // do something when the button is clicked
    }
}
```


REGISTERING EVENTS IN LAYOUT.XML FILE

You can also register event-handler method in your layout xml files

- Using xml attribute to name the handler method
- Handler method can be named anything, but needs to be public, accept a View parameter, and return nothing

```
<!-- In your layout xml file -->
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tap Me"
    android:onClick="buttonClicked" />
```

```
// In your Activity
```

```
public void buttonClicked(View view)
{
    System.out.println("clicked");
}
```

COMMON LAYOUT OBJECTS

ConstraintLayout

- The latest layout in Android for building responsive UI supporting multiple screen types

LinearLayout

- aligns all children in a single direction — vertically or horizontally

TableLayout

- positions its children into rows and columns.
- does not display border lines for their rows, columns, or cells

AbsoluteLayout

- enables child views to specify their own exact x/y coordinates on the screen
- Coordinates (0,0) is the upper left corner

RelativeLayout

- child views specify their position relative to the parent view or to each other
- E.g. two elements by right border, or make one below another, centered in the screen...

LAB — DEFINE THE UI

Create a simple user interface for Android application

There should be a UI for a weather application that we'll develop during the course

More details from the instructor

RESOURCES

Resources are an integral part of Android applications

- Application design is clearer – separate logic from definitions
- Internationalization becomes easier

Available resource types

- Strings and styled text
- Color values
- Drawables (i.e. icons, colored areas)
- Simple animations
- Menus
- Layout
- Styles and themes

RESOURCE FOLDERS BY TYPE

res/drawable/ - bitmaps & others types those can be drawn

res/layout/ - UI layout declarations

res/values/ - simple resources, typical files in this folder:

- **colors.xml** to define color drawables and color string values
- **strings.xml** to define string values
- **styles.xml** to define style objects

res/xml/ - arbitrary XML files

res/mipmap/ - bitmaps for different display types

res/raw/ - arbitrary files to be added uncompiled to installation package

USING A STRING RESOURCE

Simple values, like Strings, are defined in **res/values/** folder of the project

In **strings.xml**, define all required strings

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World</string>
    <string name="app_name">Calculator</string>
    <string name="esc_chars">This'll also work</string>
</resources>
```

// reference from java

```
CharSequence str = getString(R.string.hello);
TextView tv = (TextView)findViewById(R.id.text);
tv.setText(str);
```

<!-- Reference from XML -->

```
<TextView android:layout_width="fill_parent" android:layout_height="wrap_content"
android:textAlign="center" android:text="@string/hello"/>
```

LOCALIZING WITH RESOURCES

- create parallel resource folders with qualifiers appended to the folder names

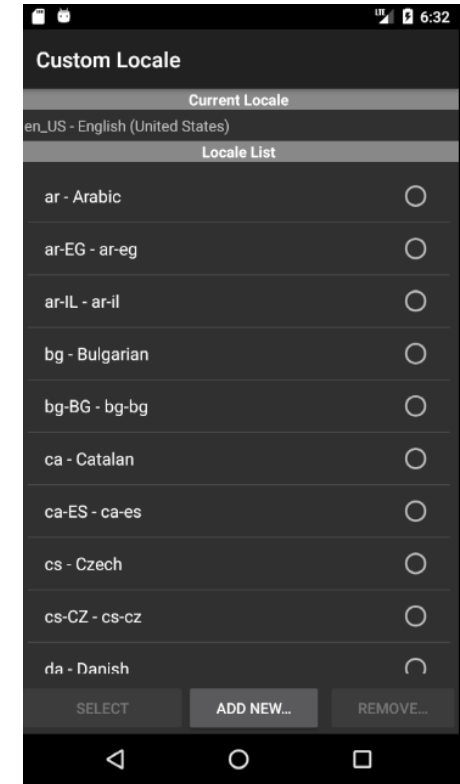
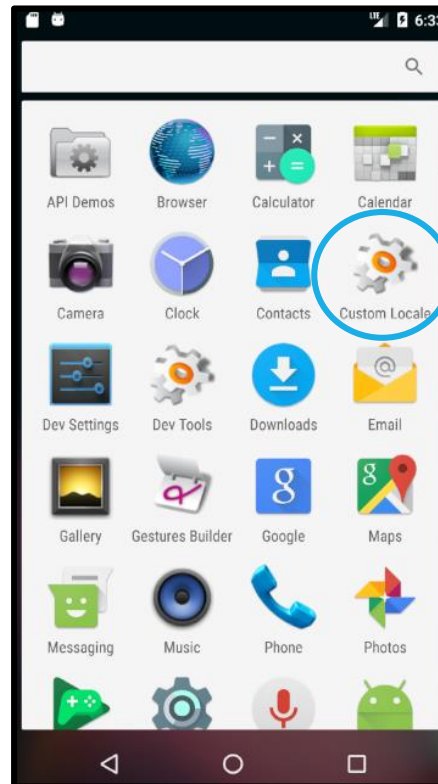
```
MyApp/  
res/  
    values-fi/  
        strings.xml    // Finnish locale  
    values-en/  
        strings.xml    // English locale
```

- Language code follows the two letter ISO 639-1 language code in lowercase. For example: en, fr, es
- Other qualifiers can be used and chained:

```
MyApp/  
res/  
    drawable-fi-finger/  
    drawable-port/  
    drawable-port-160dpi/  
    drawable-qwerty/
```

TESTING LOCALE IN ANDROID EMULATOR

- You can test localized apps against different locales with Custom Locale app found in Android Emulator



	Low density (120), <i>ldpi</i>	Medium density (160), <i>mdpi</i>	High density (240), <i>hdpi</i>	Extra high density (320), <i>xhdpi</i>
Small screen	QVGA (240x320)		480x640	
Normal screen	WQVGA400 (240x400) WQVGA (240x432)	HVGA (320x480)	WVGA (480x800) WVGA854 (480x854) 600x1024	640x960
Large screen	WVGA800 (480x800) WVGA854 (480x854)	WVGA800 (480x800) WVGA854 (480x854) 600x1024		
Extra large screen	1024x600	WXGA (1280x800) 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600

Density Bucket Name	Density Bucket	Screen Density	Ratio	Scaler
Low	ldpi	120 dpi	3	0.75
Medium	mdpi	160 dpi	4	1.00
TV	tvdpi	213 dpi	5.325	1.33
High	hdpi	240 dpi	6	1.50
Extra High	xhdpi	320 dpi	8	2.00
Extra Extra High	xxhdpi	480 dpi	12	3.00

LAB — ADDING LOCALIZATION SUPPORT

- Localize your application for at least 2 different languages of your choice and test it in your device or Android emulator

CONSTRAINTLAYOUT

With ConstraintLayout we can create large and complex layouts with flat view hierarchy

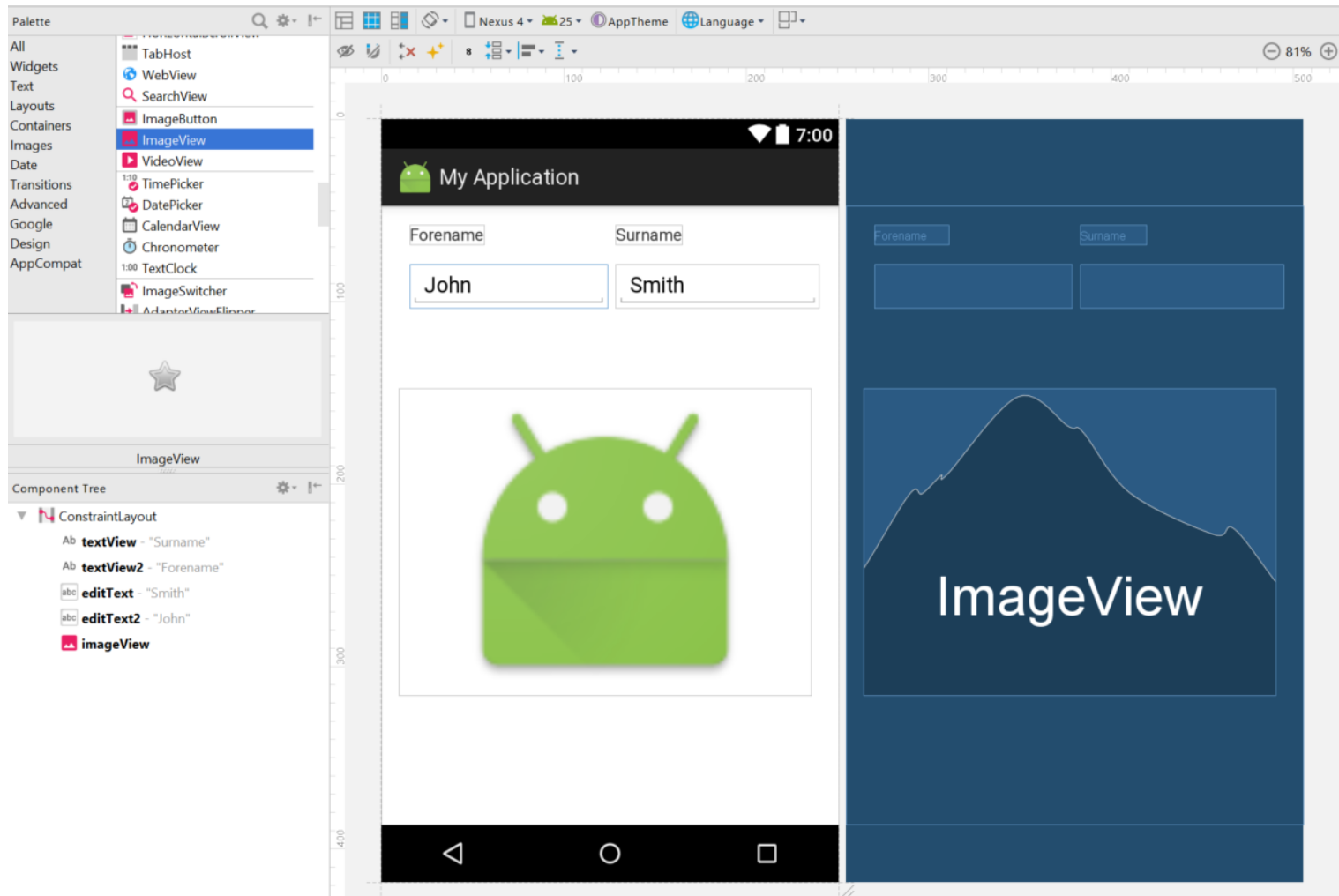
Each view in ConstraintLayout can specify relationships with other views on the layout

Can easily be used with Android Studio's layout editor

More info and tutorials at:

<https://developer.android.com/training/constraint-layout/index.html>

CONSTRAINTLAYOUT



ACTIVITIES AND INTENTS

COMPONENT LIFECYCLE

All components do have a lifecycle

An Activity has three states:

It is *active* or *running* when it is in the foreground of the screen

It is *paused* if it has lost focus but is still visible to the user

It is *stopped* if it is completely obscured by another activity. It still retains all state and member information

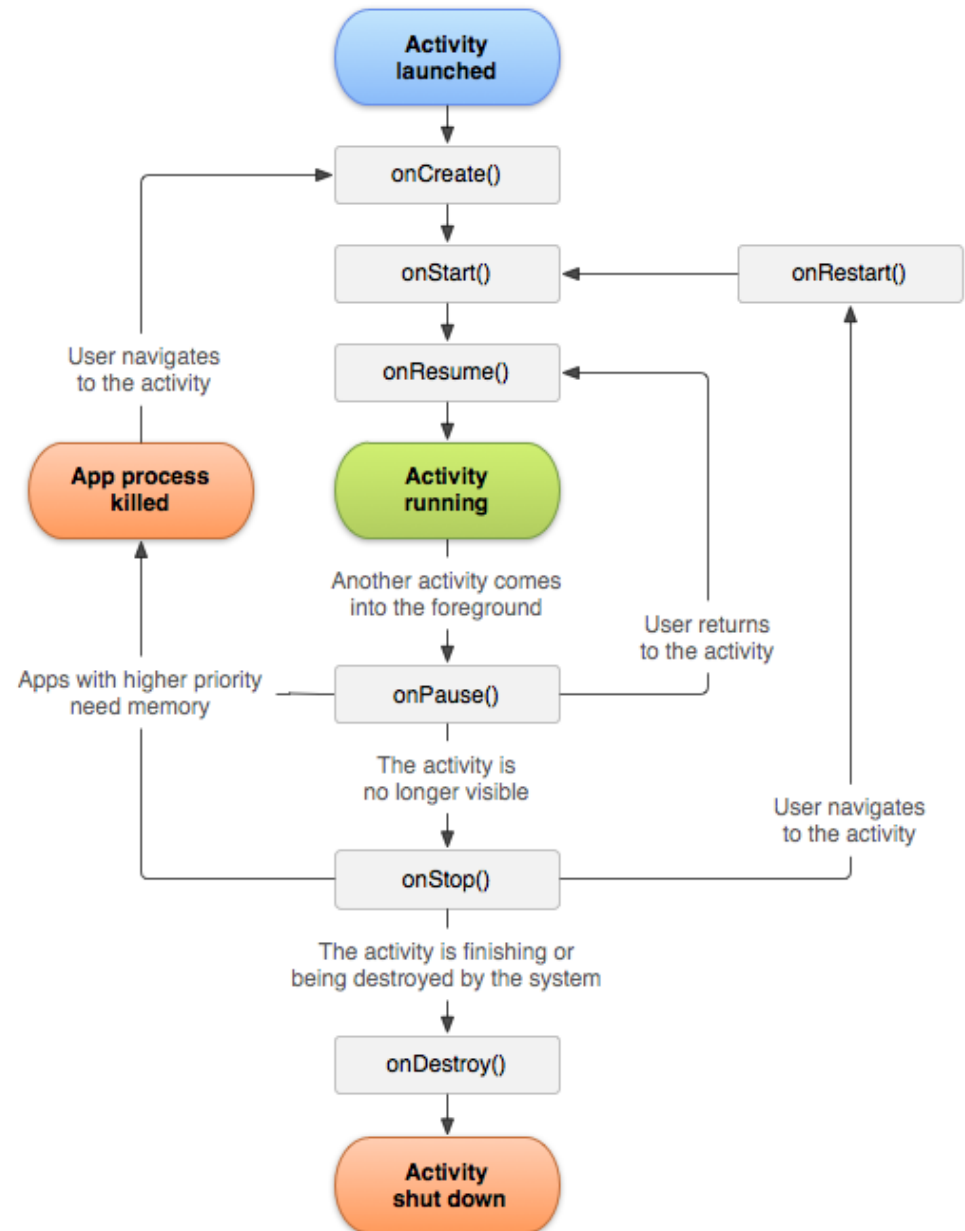
State transition diagram can be found from Android documentation:

<https://developer.android.com/guide/components/activities/intro-activities>

ACTIVITY EXAMPLE

```
public class MyActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); // Activity is being created.
    }
    protected void onStart() {
        super.onStart(); // Activity is about to become visible.
    }
    protected void onResume() {
        super.onResume(); // Activity has become visible
    }
    protected void onPause() {
        super.onPause(); // Another activity is taking focus
    }
    protected void onStop() {
        super.onStop(); // Activity is no longer visible
    }
    protected void onDestroy() {
        super.onDestroy(); // The activity is about to be destroyed.
    }
}
```

ACTIVITY STATES



SO, HOW DO YOU IMPLEMENT AN ACTIVITY?

Create a public class that extends **android.app.Activity**

- As always, class should be public, with no-args constructor, and it should not be abstract/final

Implement suitable life-cycle methods to make it react to phases

- At minimum, you would implement *onCreate()* to show something on screen
- Implementing *onPause()/onResume()* would let you turn on/off services to consume resources and to avoid annoying user

Activity will typically load a UI template and inflate it to make it visible

WHEN DOES ANDROID KILL YOUR ACTIVITY?

Android system is allowed to kill your application process or destroy your activity to free memory/other resources if necessary

- Android applications don't typically have 'quit' or 'exit' buttons
- Applications are in the memory with other apps, killed when resources are needed elsewhere, and reactivated when user wants to use them
- Killing process won't automatically remove object from memory, but destroying application object will clean everything

APPLICATION LIFE-CYCLE

Application can be killed anytime after `onPause()`, `onStop()`, or `onDestroy()` is called

- But not after `onCreate()`, `onStart()`, `onResume()`, or `onRestart()`
- **`onPause()` is good place to write crucial persistent data to storage** – since `onStop()/onDestroy()` might not be called
- When application is used again, Android will call `onCreate()`, `onStart()`, and `onResume()` again

Android will destroy your application if memory is running short

- It can still be re-started later when needed, it's just heavier life-cycle than just killing the application process

SURVIVING DESTRUCTION

Android may destroy your application, removing it from memory if needed

Android will also destroy and restart your app when screen orientation changes

If you want transient variable data (member variables) to survive this, you should implement one more life-cycle method

onSaveInstanceState()

- It is called before destroying the application
- You get a Bundle object that you can use to store information that should survive
- Use methods such as `putString()` and `putInt()` to store variable state
- When application is recreated, both `onCreate()` and `onRestoreInstanceState()` life-cycle methods will receive same bundle and can use the data in it
 - If there's no state information to restore, bundle is *null*
- **Remember to always call `super()` to also store state on View components**

EXAMPLES ON ONSAVEINSTANCESTATE()

@Override

```
public void onSaveInstanceState(Bundle savedInstanceState) {  
    super.onSaveInstanceState(savedInstanceState);  
  
    // Save UI state changes to the savedInstanceState. This bundle  
    // will be passed to onCreate if the process is killed and restarted.  
  
    savedInstanceState.putDouble("myDoubleValue", 2.04);  
  
    savedInstanceState.putInt("myIntegerValue", 1);  
  
    savedInstanceState.putString("myStringValue", "Hello World!");  
  
    // etc.  
}
```

@Override

```
public void onRestoreInstanceState(Bundle savedInstanceState) {  
    super.onRestoreInstanceState(savedInstanceState);  
  
    // Restore UI state from the savedInstanceState.  
    // This bundle has also been passed to onCreate.  
  
    double myDouble = savedInstanceState.getDouble("myDoubleValue");  
  
    int myInt = savedInstanceState.getInt("myIntegerValue");  
  
    String myString = savedInstanceState.getString("myStringValue");  
}
```

WHAT HAPPENS WHEN YOU NAVIGATE FROM ACTIVITY A TO ACTIVITY B?

Activity A's onPause() method executes.

Activity B's onCreate(), onStart(), and onResume() methods execute in sequence. (Activity B now has user focus.)

Then, if Activity A is no longer visible on screen, its onStop() method executes.

LAB: ACTIVITY LIFE-CYCLE AND STATE

Implement state as member variables and observe how it behaves when screen orientation changes

Fix the problems using bundle and onSaveInstanceState() method

INTENTS

INTENTS - MESSAGES

Intent is more or less like an *asynchronous message* from one part of application to another

- Intent Filter tells that who can handle the message

Intent can be sent to activate another View or Service

- It can also contain data
- So basically, it's a call to activate something + data to send with the call

It may have exactly one recipient, or it may have multiple potential recipients – but only one will be chosen to run at any time

- Tight coupling vs loose coupling – explicit vs implicit intent resolution

INTENTS

Activity 1

I *INTEND* to navigate to another view

I *INTEND* to open an web url address

I *INTEND* to load View2.class next

I *INTEND* to show a picture

Activity 2

Activity 3

Service

STRUCTURE OF AN INTENT

Intent can contain

- **Action:** a string naming the action to be performed
- **Data:** The URI of the data to be acted on and the MIME type of that data
- Alternatively, you can use `setComponent(componentName)` or `setClass(Context, class)` to register specific handler (explicit intent)

Additionally, it can have

- **Category:** a string containing additional information about the kind of component that should handle the intent
- **Type:** Explicit mime type of the intent data
- **Extras:** key-value pairs for additional information that should be delivered to the component handling the intent
- **Flags:** instruct the Android system how to launch an activity and how to treat it

INTENT RESOLUTION

Two main types of Intents

- **Explicit** – Intent targeted to a specific Activity, used normally only in application's internal messaging
- **Implicit** – Intent targeted to any Activity that can handle it, used to activate components in other applications

Explicit intent is delivered directly to target class

Implicit Intents are resolved by using specific Intent Filters by

- Action
- Data
- Category

If there is no handlers for intent you'll get
`android.content.ActivityNotFoundException!`

USING EXPLICIT INTENTS

Explicit intent is specific and will run just that one named intent next, along with any data you wish to pass

- Intent parameters are context (your activity), and target activity/service to invoke

```
Intent explicitIntent =  
    new Intent( this,  MySecondActivity.class);  
  
explicitIntent.putExtra(  
    "Value1", "This value one for ActivityTwo ");  
  
explicitIntent.putExtra(  
    "Value2", "This value two ActivityTwo");  
  
startActivity(explicitIntent);
```

INTENT FILTER

Explicit intents are delivered always, no filters are used

- If a component does not define Intent filters, it can only be called by explicit Intents.

Implicit intent is delivered to a component only if it can pass through one of the component's filters

- Component has separate filters for each job it can do

Intent Filters are registered to Android system inside `AndroidManifest.xml`

- System must be aware of capabilities before launching the component
- When creating new actions, use your package name as basis for name, for example `fi..feedreader.SHOW_MENU`
- Broadcast intents can be registered also in code

INTENT MATCHING

All activities that specifies Intents Filters for both

android.intent.action.MAIN

android.intent.category.LAUNCHER

are shown in application launcher

Home screen of an task is searched by looking for the activity with

android.intent.category.HOME

Use **PackageManager** to find components for a specific event

- a set of **query...()** methods that return all components that can accept a particular intent
- A series of **resolve...()** methods that determine the best component to respond to an intent

EXAMPLES OF IMPLICIT INTENTS:

```
public static void invokeWebBrowser(Activity activity) {
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse("http://www.google.com"));
    activity.startActivity(intent);
}

public static void invokeWebSearch(Activity activity) {
    Intent intent = new Intent(Intent.ACTION_WEB_SEARCH);
    intent.setData(Uri.parse("http://www.google.com"));
    activity.startActivity(intent);
}

public static void dial(Activity activity) {
    Intent intent = new Intent(Intent.ACTION_DIAL);
    activity.startActivity(intent);
}
```


EXAMPLES OF IMPLICIT INTENTS:

```
public static void call(Activity activity) {
    Intent intent = new Intent(Intent.ACTION_CALL);
    intent.setData(Uri.parse("tel:555-555-5555"));
    activity.startActivity(intent);
}

public static void showMapAtLatLng(Activity activity) {
    Intent intent = new Intent(Intent.ACTION_VIEW);
    //geo:lat,long?z=zoomlevel&q=question-string
    intent.setData(Uri.parse("geo:0,0?z=4&q=business+near+city"));
    activity.startActivity(intent);
}
```

DATA TRANSFER BETWEEN ACTIVITIES

It's possible to send some data with Intent using

```
intent.putExtra("MY_VARIABLE_KEY", variableToPass);
```

Reading on receiver side can be done with

```
Intent intent = new Intent("name.of.the.ActivityB");  
intent.putExtra("MY_VARIABLE_KEY", variableToPass);
```

Key is always a String

Value can be primitive data types (int, float,..), String, Bundle, Parcelable and Serializable.

LAB: USE INTENTS TO NAVIGATE

Create another Activity and use intent to navigate between them

We'll create a settings view and a forecast view to our weather application

LAB: INTENT FILTERING AND EXISTING SERVICES

Use intents to trigger existing services

- Open web browser to specific address, do a web search with a keyword, open dialer to dial a number

NETWORKING — HTTP/JSON

"RAW" HTTP NETWORKING

All networking should be done outside the main GUI thread of an android application

Using networking in the main thread will cause your app to be signalled out (killed)

The classical HTTP networking uses Android's AsyncTask (background thread) with HttpGetRequest. An example

<https://medium.com/@JasonCromer/android-async-task-http-request-tutorial-6b429d833e28>

However, there are better abstractions available like Volley and Retrofit libraries.

USING VOLLEY FOR HTTP NETWORKING

Volley library is one of the most used abstractions for networking

It implements the state machine for HTTP networking needed by an app

The complete guide and example:

- <https://developer.android.com/training/volley/simple.html#java>

LAB — REQUESTING AND PARSING HTTP/JSON

Let's do this in action as an example

We'll use some open weather API to get the current weather data on the screen.



USING DEVICE APIS

MODULE CONTENTS

Accelerometer, Light, Orientation etc. – connect to a system service

WiFi Manager – connect to a system service

Battery Status – register to a Broadcast Intent

HARDWARE SENSORS IN ANDROID

Android SDK provides access to raw data from sensor service via **SensorManager** object

```
SensorManager sensors = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

- Register a listener to receive events
 - Listener uses **SensorEventListener** interface (next slide)

```
boolean isAvailable = sensors.registerListener(  
    Sensors.this, // listener interface  
    sensors.getDefaultSensor(Sensor.TYPE_ORIENTATION),  
    SensorManager.SENSOR_DELAY_NORMAL);
```

SENSOREVENTLISTENER

SensorManager sends the values back to application using **SensorEventListener** interface

- Implement and override two methods in your activity (or other sensor listener class):

```
public void onAccuracyChanged(Sensor sensor, int accuracy){  
    //Called when the accuracy of a sensor has changed.  
}  
public void onSensorChanged(SensorEvent event) {  
    //Called when sensor values have changed.  
}
```

READING SENSOR VALUES

Inside **onSensorChanged()** check first which sensor sent the event

TYPE_ACCELEROMETER - A constant describing an accelerometer sensor type.

TYPE_ALL - A constant describing all sensor types.

TYPE_GYROSCOPE - A constant describing a gyroscope sensor type

TYPE_LIGHT - A constant describing an light sensor type.

TYPE_MAGNETIC_FIELD - A constant describing a magnetic field sensor type.

TYPE_ORIENTATION - A constant describing an orientation sensor type.

TYPE_PRESSURE - A constant describing a pressure sensor type

TYPE_PROXIMITY - A constant describing an proximity sensor type.

TYPE_TEMPERATURE - A constant describing a temperature sensor type

SensorEvent contains the values of the sensor

- See below

READING THE ACCELEROMETER

The accelerometer values are defined in the accelerometer reference:

- Sensor values are acceleration in the X, Y and Z axis, where the X axis has positive direction toward the right side of the device, the Y axis has positive direction toward the top of the device and the Z axis has positive direction toward the front of the device.
- The direction of the force of gravity is indicated by acceleration values in the X, Y and Z axes.
- The typical case where the device is flat relative to the surface of the Earth appears as `-STANDARD_GRAVITY` in the Z axis and X and Y values close to zero.
- Acceleration values are given in SI units (m/s^2).

READING THE ORIENTATION

The orientation sensor values are defined in the orientation sensor reference


- Sensor values are yaw, pitch and roll Yaw is the compass heading in degrees, range $[0, 360[$ 0 = North, 90 = East, 180 = South, 270 = West Pitch indicates the tilt of the top of the device, with range -90 to 90.
- Positive values indicate that the bottom of the device is tilted up and negative values indicate the top of the device is tilted up.
- Roll indicates the side to side tilt of the device, with range -180 to 180.
- Positive values indicate that the left side of the device is tilted up and negative values indicate the right side of the device is tilted up.

SIMULATING BATTERY STATUS WITH ANDROID EMULATOR

Extended controls

 Location

 Cellular

 Battery

 Phone

 Directional pad

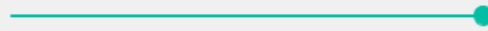
 Fingerprint

 Virtual sensors

 Settings

 Help

Charge level

 100%

Battery health

Good

Charger connection

AC charger

Battery status

Charging

Extended controls

- Location
- Cellular
- Battery
- Phone
- Directional pad
- Fingerprint
- Virtual sensors

Settings

Help

Accelerometer Additional sensors



☒ Rotate ☐ Move

Yaw 0.0

Pitch 0.0

Roll 0.0

Device rotation



Resulting values

Accelerometer (m/s^2): 0.00 9.81 0.00
Magnetometer (μT): 22.00 5.90 43.10
Rotation: ROTATION_0

Extended controls

Location

Cellular

Battery

Phone

Directional pad

Fingerprint

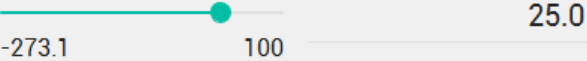
Virtual sensors

Settings

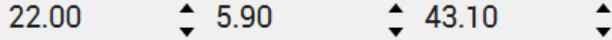
Help

Accelerometer Additional sensors

Ambient temperature (°C) ?



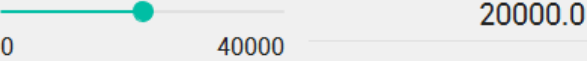
Magnetic field (μT) ?



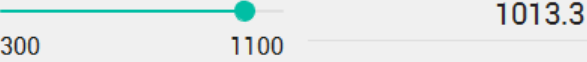
Proximity (cm) ?



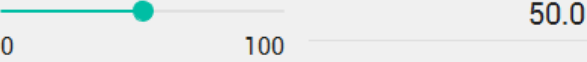
Light (lux) ?



Pressure (hPa) ?



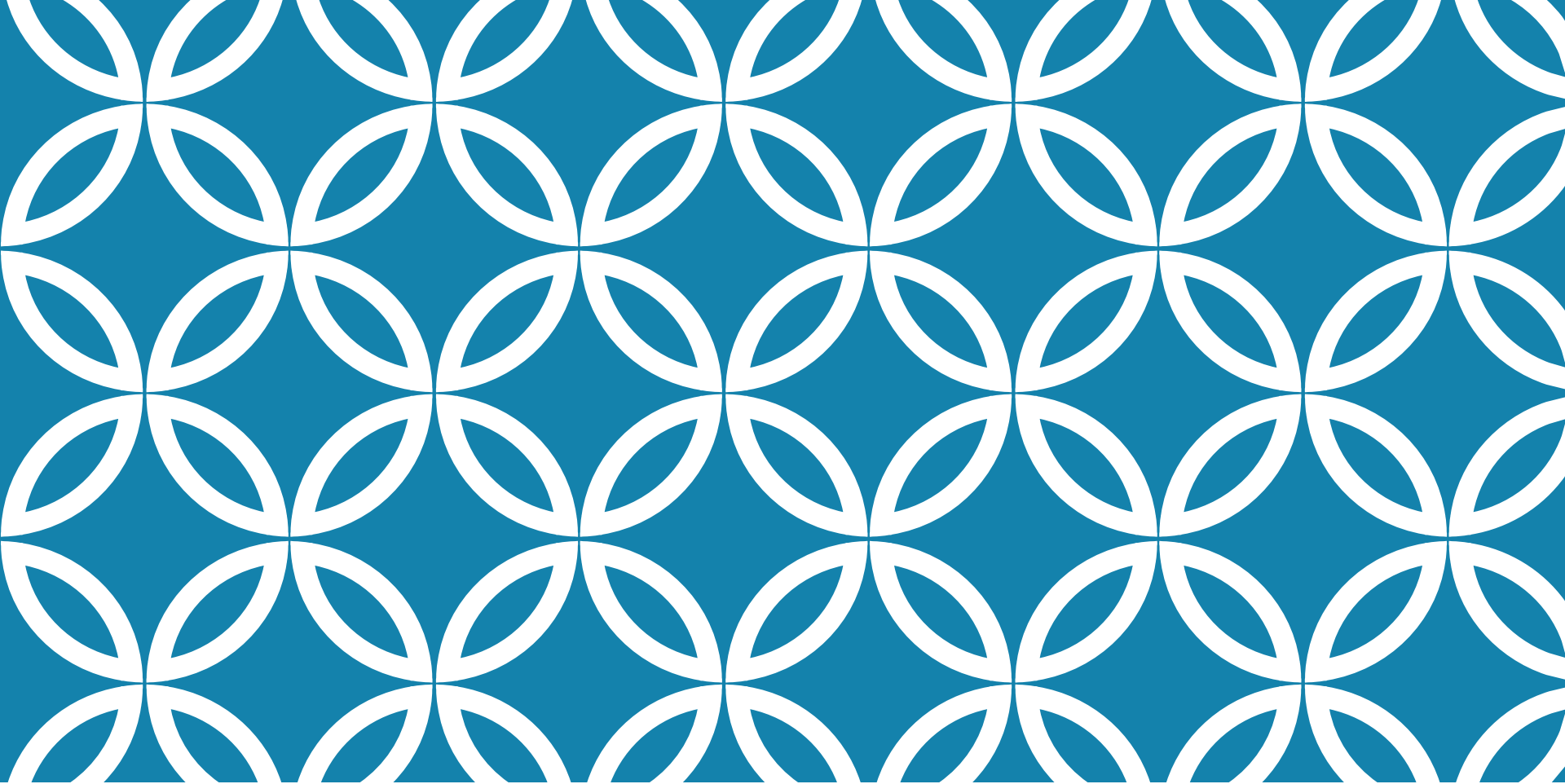
Relative humidity (%) ?



LAB — SENSORS

Create an Android app and write code that will use different sensors

If you can't test on actual hardware, use `SensorSimulator` instead



APPLICATION SECURITY

SECURITY AND PERMISSIONS

Android is a multi-process system

- each application (and parts of the system) runs in their own processes
- Most security between applications and the system is enforced at the process level through standard Linux facilities, such as user and group IDs that are assigned to applications
- Each Android package (.apk) file installed on the device is given its own unique Linux user ID, creating a sandbox for it and preventing it from touching other applications (or other applications from touching it). This user ID is assigned to it when the application is installed on the device, and remains constant for the duration of its life on that device.

Additional security features

- A "permission" mechanism that enforces restrictions on the specific operations that a particular process can perform
- Per-URI permissions for granting ad-hoc access to specific pieces of data.

SECURITY ARCHITECTURE

By default, no application has permission to perform any operations that would adversely impact other applications, the operating system, or the user

- This includes reading or writing the user's private data (such as contacts or e-mails), reading or writing another application's files, performing network access, keeping the device awake, etc.

An application's process is a secure sandbox

- It can't disrupt other applications

If an application needs for additional capabilities not provided by the basic sandbox, it must explicitly declare the *permissions*

- These permissions it requests can be handled by the OS in various ways
 - By automatically allowing or disallowing based on certificates or by prompting the user
 - Permissions declared statically in that application, so they can be known up-front at install time and will not change after that.

REQUESTING PERMISSIONS AT RUN TIME

From Android 6.0 (API 23), permissions are divided into dangerous and normal permissions

All permissions are listed in project's manifest file but dangerous permissions are granted by the user at run-time

Before Android 6.0, all permissions were granted by user at installation phase only

Operations with dangerous permissions deal somehow with user's privacy or private data

DANGEROUS PERMISSIONS

Permission Group	Permissions
CALENDAR	<ul style="list-style-type: none">•READ_CALENDAR•WRITE_CALENDAR
CAMERA	<ul style="list-style-type: none">•CAMERA
CONTACTS	<ul style="list-style-type: none">•READ_CONTACTS•WRITE_CONTACTS•GET_ACCOUNTS
LOCATION	<ul style="list-style-type: none">•ACCESS_FINE_LOCATION•ACCESS_COARSE_LOCATION
MICROPHONE	<ul style="list-style-type: none">•RECORD_AUDIO
PHONE	<ul style="list-style-type: none">•READ_PHONE_STATE•CALL_PHONE•READ_CALL_LOG•WRITE_CALL_LOG•ADD_VOICEMAIL•USE_SIP•PROCESS_OUTGOING_CALLS
SENSORS	<ul style="list-style-type: none">•BODY_SENSORS
SMS	<ul style="list-style-type: none">•SEND_SMS•RECEIVE_SMS•READ_SMS•RECEIVE_WAP_PUSH•RECEIVE_MMS
STORAGE	<ul style="list-style-type: none">•READ_EXTERNAL_STORAGE•WRITE_EXTERNAL_STORAGE

HANDLING RUN-TIME PERMISSIONS IN CODE

Sensitive operations with dangerous permissions should always be checked (and permission granted) at run-time

Sensitive API call without permission causes `SecurityException`

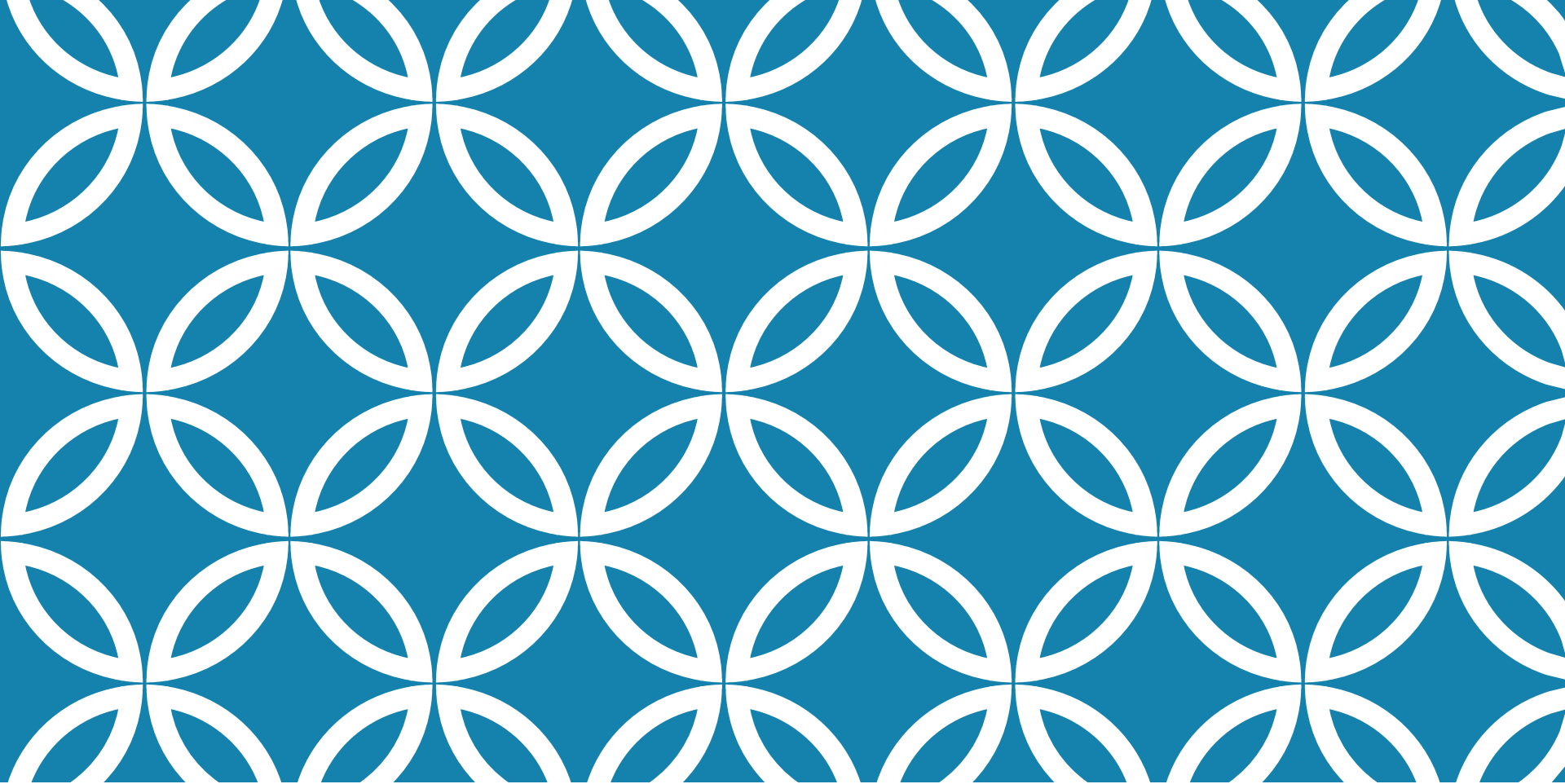
Run-time check applies to Android 6.0 (API level 23) onwards

The version can also be checked run-time:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {  
    if (checkSelfPermission(Manifest.permission.READ_CALL_LOG) !=  
        PackageManager.PERMISSION_GRANTED) {  
        ActivityCompat.requestPermissions(this,  
            new String[]{Manifest.permission.READ_CALL_LOG}, 0);  
        return;  
    }  
}
```

HANDLING RUN-TIME PERMISSIONS IN CODE

<https://developer.android.com/training/permissions/requesting#kotlin>



LOCATION API

INTRODUCTION TO LOCATION BASED SERVICES

Location information is almost a requirement for cell phones these days

Several different ideas already seen for applications make the feature exciting

- Not only navigation but social applications, security, outdoor activities...
- What is not yet invented?

Google has made a large effort on creating not only world wide maps

- Satellite images covering almost the whole globe
- Traffic information
- Street View
- Navigation (currently only in the USA)
- Latitude service

USING THE GLOBAL POSITIONING SERVICES (GPS)

Android SDK provides a way to use GPS as part of application

Optional way of getting more coarse location data exist

- Using WiFi and GSM location
- Can be battery consuming and slower

To get location information via system service **LocationManager**

- Create a **LocationManager** object
- Retrieve an object instance to it from system (**getSystemService()**)
- Add required permissions to manifest
- Get the currently best location provider from manager by using **getBestProvider()** method
- Use the data

CONNECTING TO LOCATIONMANAGER

To get a **LocationManager** instance

```
LocationManager lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE)
```

- Request permissions in **AndroidManifest.xml**

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

PROVIDE A LOCATIONLISTENER

Before the provider returned by `getBestProvider` can be used, a `LocationListener` needs to be implemented

- Implement four methods:

```
void onLocationChanged(Location location)
```

```
void onProviderDisabled(String provider)
```

```
void onProviderEnabled(String provider)
```

```
void onStatusChanged( String provider, int status, Bundle extras)
```

THE LOCATION OBJECT

Received by the listener, contains the most recent location

Consists of a latitude and longitude, a UTC timestamp.

- optionally information on altitude, speed, and bearing

Information may be specific to a particular provider or class of providers

- communicated to the application using `getExtras`
- returns a Bundle of key/value pairs
- Each provider will only provide those entries for which information is available

Has several methods to retrieve location data, or e.g. get a distance moved after last call to **`onLocationChanged()`**

REQUEST FOR LOCATION UPDATES

The complete example:

- <https://developer.android.com/training/location/receive-location-updates>


LAB — LOCATION APIS

Write some code to try out location APIs


Implement location feature to the weather application so that the application shows the weather (and forecast) based on user's location instead of cityname.

TESTING LOCATION WITH EMULATOR

Extended controls

 Location

 Cellular

 Battery

 Phone

 Directional pad

 Fingerprint

 Virtual sensors

 Settings

 Help

GPS data point

Coordinate system

Decimal

Longitude

-122.084

Currently reported location

Longitude: -122.0840
Latitude: 37.4220
Altitude: 0.0

Latitude

37.422

Altitude (meters)

0.0

 SEND

GPS data playback

Delay (sec)	Latitude	Longitude	Elevation	Name	Description



Speed 1X

 LOAD GPX/KML