CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

# DIT341 - Lecture 9:
## Mobile Development with Android

Gregory Gay
(Some slides by Grischa Liebel)

# This Lecture

- Android OS, SDK, and Runtime
- Basics of native Android programming
  - Basic App Components
    - Activities and Intents
  - Manifest and Resources
  - Permissions
  - Layouts and Responsive UI Design
  - Making REST requests

# Related Activities

- **Activity 20: Recommended**
- Practice Android with Codelabs for Android Developer Fundamentals (V2):
  - https://developer.android.com/courses/fundamentals-training/toc-v2
  - 1.2 Part A: Your first interactive UI
  - 1.3: Text and scrolling views
  - 2.1: Activities and intents

# What is a Mobile Application?

- "A mobile app or mobile application is a computer program designed to run on a mobile device such as a phone/tablet or watch." - Wikipedia
- Isn’t a mobile device just a smaller computer?
  - Mobility imposes restrictions on program design.
    - Computational Power (small devices, limited power)
    - Battery (must last for as long as possible)
    - Input Methods (may be touch only)
    - Screen Size (watch, small screen)
    - Unreliable Network Connection (can’t assume constant connection)

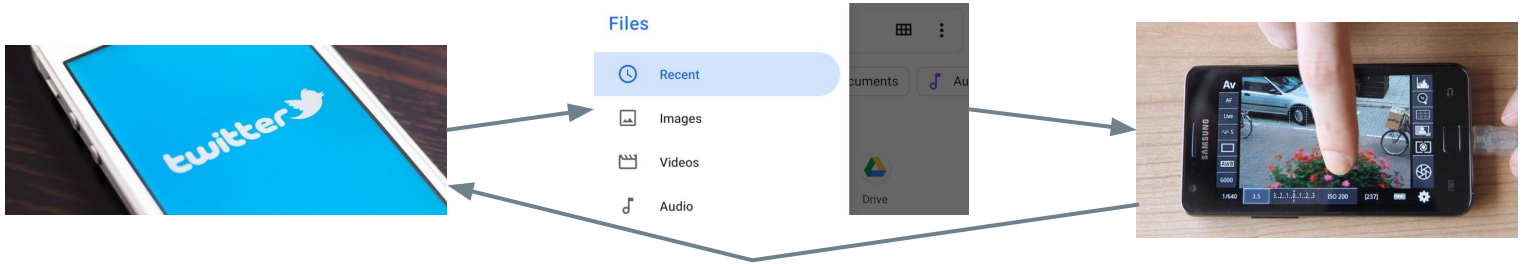
# Differences from “Regular” Apps

- Not strictly defined, but…
- Mobile apps must adapt to multiple devices.
  - Different layouts for different screen sizes.
  - Can enable/disable features based on hardware.
  - Requires more thought about design and UX.



# Differences from “Regular” Apps

- Apps are designed to form an ecosystem.
  - Each app is a collection of components:
    - Activities provide a user interface.
      - The main activity starts when you tap the icon, but other apps can directly link to other “activities”.
    - Broadcast receivers and Services perform background tasks.
  - Apps often specialized for smaller tasks.
    - Purpose-built apps link to other apps for common features.



# Android

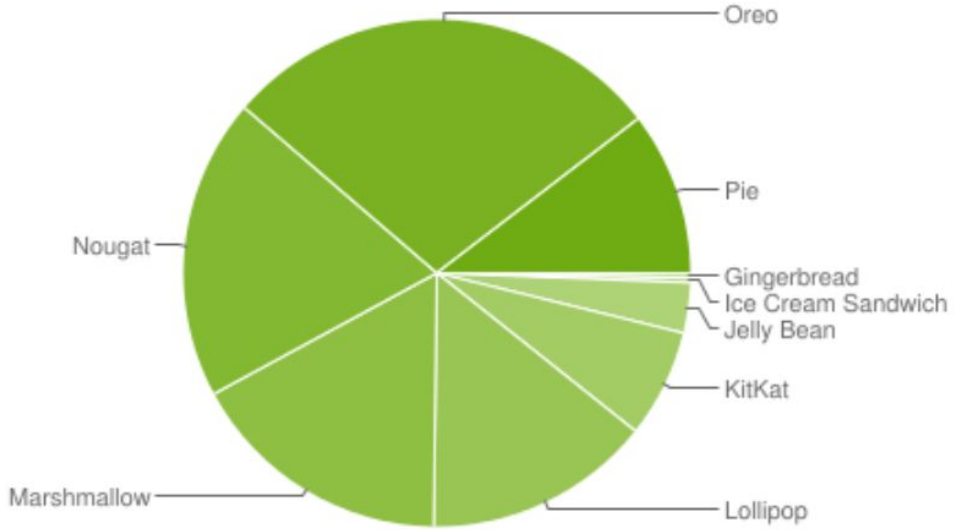
- Most popular mobile operating system
  - 2.5 **billion** active devices
  - Phones, tablets, watches, car displays, TVs, IoT devices, speakers, home automation, ...
- Apps written in Android SDK.
  - Supports Java (this course), Kotlin, C++.
- Code, data, resources compiled into APK.
  - Android Package
  - Compiled into device-specific code and run in Android OS (Linux-based).

# Android Fundamentals

- Each app operates in a sandbox.
  - Each app is a “user”.
  - Only that app can access its files.
  - Each app runs in its own virtual machine and process.
- Android implements “principle of least privilege”.
  - Each app can access only components it requires.
  - Users must grant access to location, camera, bluetooth, files, etc.

# Versioning

- Apps target a minimal “API level”.
  - Defines available features.
- Apps must update compatibility or be disabled.




*Data collected during a 7-day period ending on May 7, 2019*

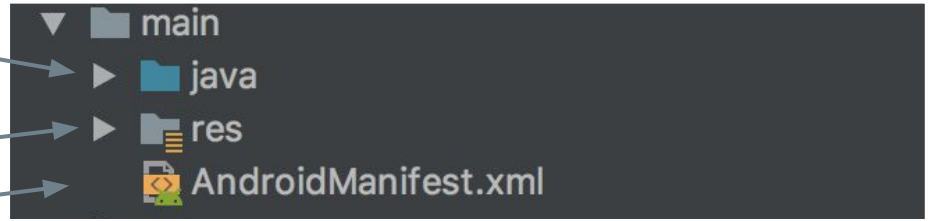
*Any versions with less than 0.1% distribution are not shown.*

| | | | | |
|---|---|---|---|---|
| Oreo[16] | 8.0 – 8.1 | 4.10 | August 21, 2017 | 26 – 27 |
| Pie[17] | **9.0** | 4.4.107, 4.9.84, and 4.14.42 | August 6, 2018 | 28 |
| **Legend:** Old version | Older version, still supported | **Latest version** | | |

# Android Fundamentals

# Android Apps - Structure

- Android Code
  - All the logic
- Resources
- Manifest



# App Components

- Core building blocks of an app.
- Entry points for an app.
- **Activities** are “screens” with a UI.
- **Services** run a task in the background.
- **Broadcast receivers** deliver events to apps outside of regular user flow.
- **Content providers** manage a pool of information.

# Activities

- A single screen with a user interface.
  - Most apps have multiple independent activities.
    - E-mail: Show messages, compose, read.
- If allowed, other apps can start any activity.
  - Camera app opens “Compose” activity to share photo.
- Activities control and link processes.
  - Ensure the current process is not killed.
  - Link to calling activities and maintain their process.
  - Model state in case process is killed.
  - Model flow between apps.

# Services

- Entry point for running a background process.
  - Playing music, sending files.
- Does not provide a direct UI.
- Can be started by an activity.
  - Can maintain a notification to allow user interaction.
  - Services without notifications can be killed if resources are needed by OS.
- Bound services offer an API to the calling app.
  - Maintained as long as needed, then killed.

# Broadcast Receivers

- Allows OS to deliver events when the app is not running.
  - Listen to system-wide broadcast announcements.
- Respond to events like a photo being taken.
- Often notify users that an event has occurred.
- Often minimal, used as a gateway to launch activities or services.

# Content Providers

- Manages a shared set of app data.
- Other, allowed, apps can query or modify the data.
  - Android has a Content Provider for contact data.
- An entry point into an app for publishing data items.
  - Identified by a URI.
  - Owning app wakes up when a URI is accessed.
  - URIs provide a secure way to pass content.
    - Content is locked and accessed through temporary permission.

# Intents

- Asynchronous messages that bind components at runtime.
  - Messengers that request actions from other components.
  - Start an activity, start a service, deliver a broadcast.
  - Can convey a result back to the caller.
- **Explicit** intents activate a specific component.
- **Implicit** intents activate types of components.

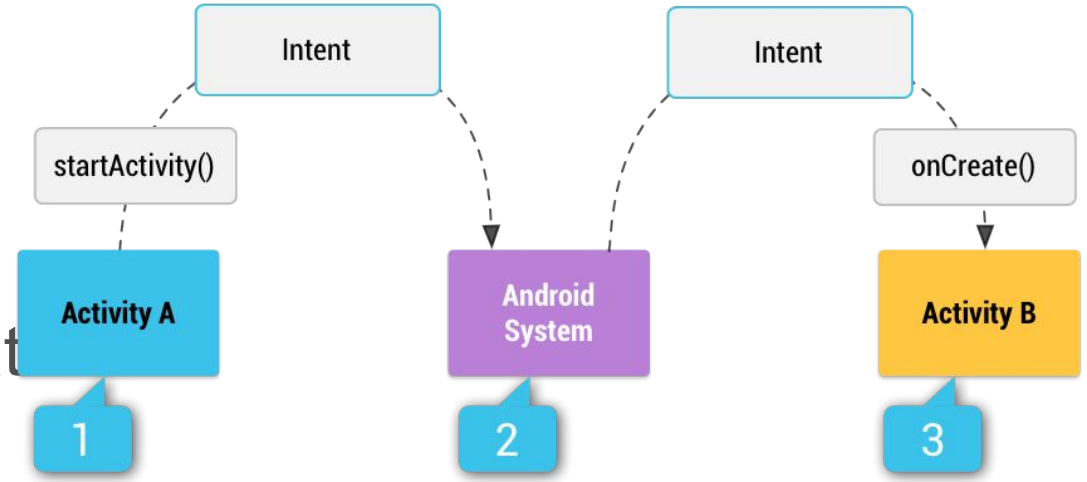
# Explicit Intents

- Name a specific app (by package or component).
- Often used by one component to start another within the same app.

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

# Implicit Intents

- Describe a type of action you want to perform.
- Allow the system to find components that can perform that action (selected by the user).
  - Done using `IntentFilters`



# Implicit Intents

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

This is a SEND action

It sends a text message

Android will search for all Activities that can handle a SEND action on plain text.

# The Manifest

- Each android app needs an `AndroidManifest.xml` file
- Essential information
  - App name
  - Components
  - SDK version
  - Permissions
  - ...

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="myapp.wm1819.grischaliebel.de.myapplication">

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme"
        android:usesCleartextTraffic="true"
        tools:ignore="GoogleAppIndexingWarning">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

# Declaring Components

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

Resource for an icon.

Class name of Activity

Declares an Activity

Label used to identify Activity

# Declaring Component Capabilities

```
<manifest ... >
    ...
    <application ... >
        <activity android:name="com.example.project.ComposeEmailActivity">
            <intent-filter>
                <action android:name="android.intent.action.SEND" />
                <data android:type="*/*" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Intent Filters declare how app responds to Intents

This Activity is registered as an option for ACTION_SEND intents.

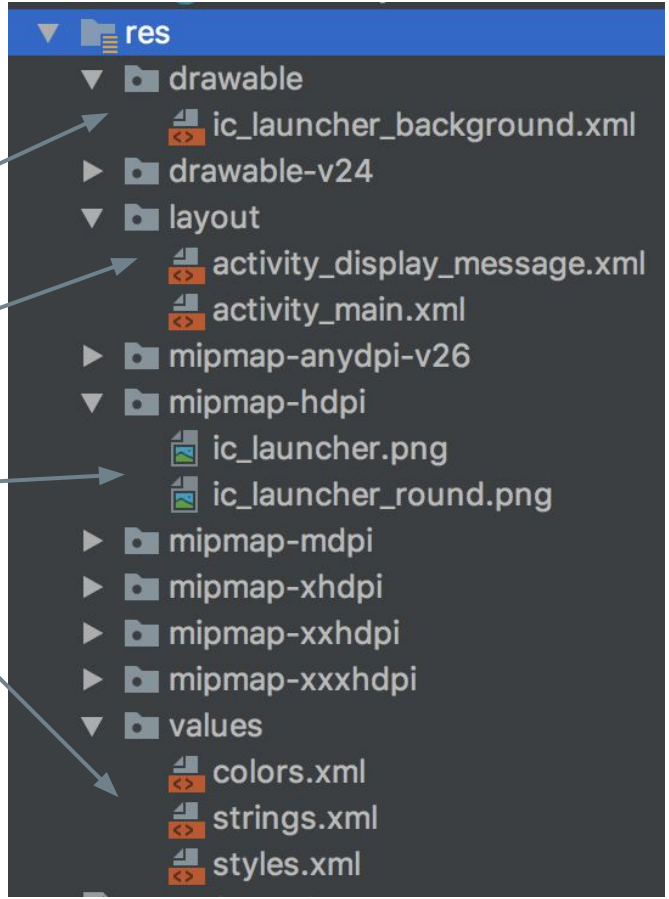
# Declaring App Requirements

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera.any"
                  android:required="true" />
    <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="19" />
    ...
</manifest>
```

- App requires at least Android 2.1 and a camera.
- Setting "`required`" attribute to false indicates that the app uses the camera, but can function without.

# App Resources

- Bitmaps/Pictures
- UI definitions in XML
- Launcher icons
- Text strings (incl. translations)



# App Resources

- Each resource is assigned a unique ID.
  - Used to reference the resource from code or layout XML.
  - File `res/drawable/logo.png` -> ID `R.drawable.logo`
- Can provide alternate resources for configurations.
  - UI strings can be used to swap one language for another
  - Qualifier appended to directory (`res/values-fr/`)
  - Many default qualifiers supported for different screen sizes, device types, orientations (layout vs portrait).
  - **Allows automated responsiveness.**

# Permissions

- By default, apps are not allowed to use hardware, access data, access network.
- Permissions must be explicitly asked for in manifest

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

- Users must grant permission for dangerous requests (like the ones above).
  - Formerly, user had to agree to all requests to install app (< API 6.0 (23)).
  - Now, permissions granted individually.

# Let’s Take a Break

# Demonstration - “My First App”

Want to work along? This demonstration follows:
https://developer.android.com/training/basics/firstapp

You will need to install Android Studio, at least one emulator, and download the sample project.

This demonstration follows sections:

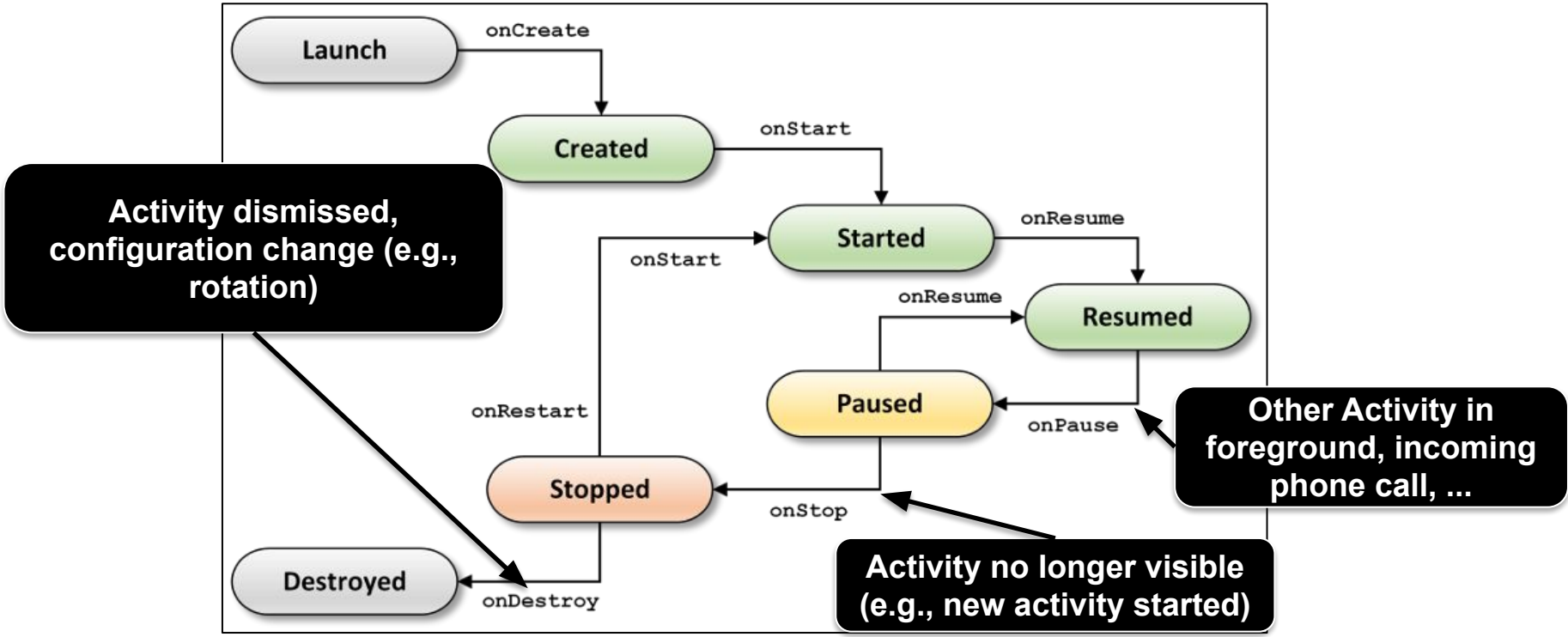
- Create an Android project
- Run your app

# Designing Activities

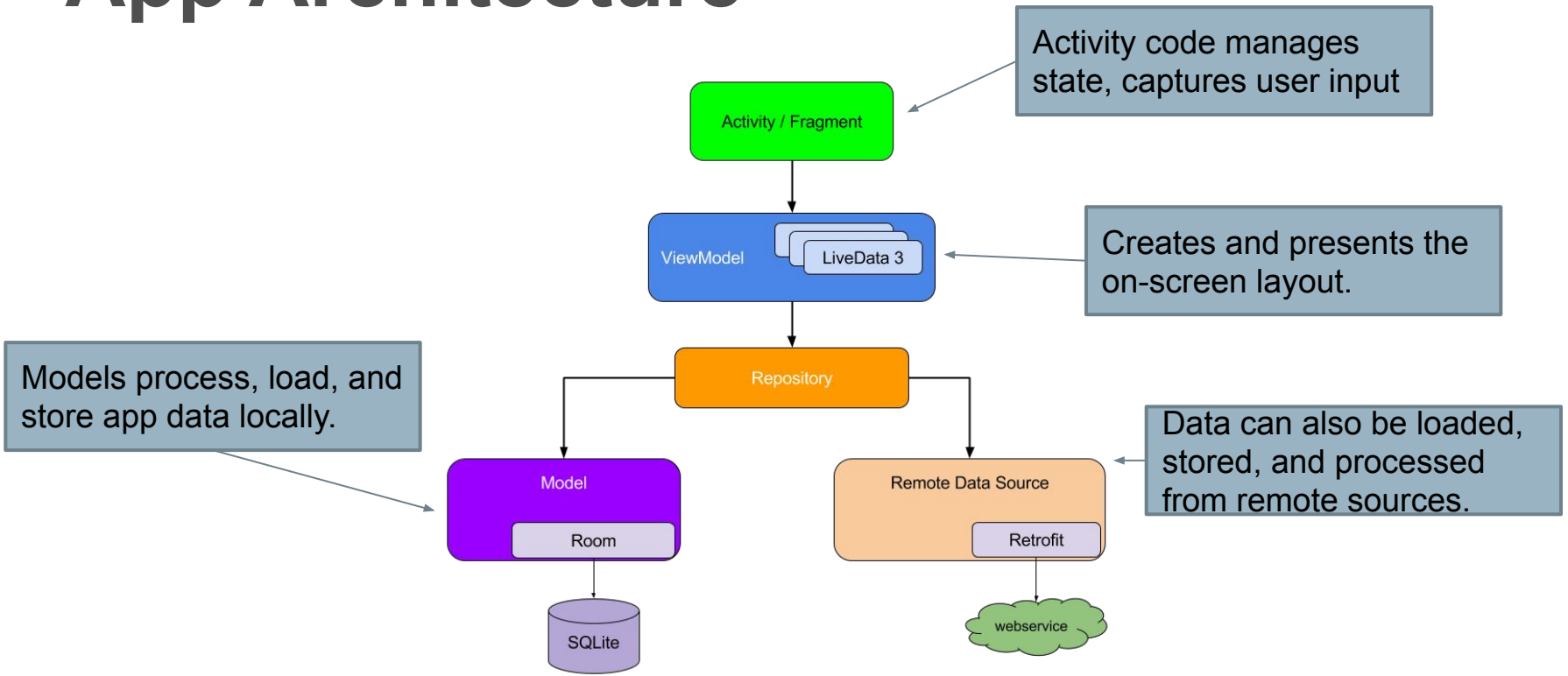
# The Concept of Activities

- Rather than interacting with apps as atomic units, Activities interact directly with Activities.
  - Any Activity can serve as an entry point to app interaction
- An Activity has a UI, and is usually a single screen.
  - An app may have a Settings Activity, a Select Photo Activity, ...
- One Activity is the “main activity”.
  - Launches when you click the icon.
- Activities must have minimal dependencies on other Activities.

# Activity Lifecycle

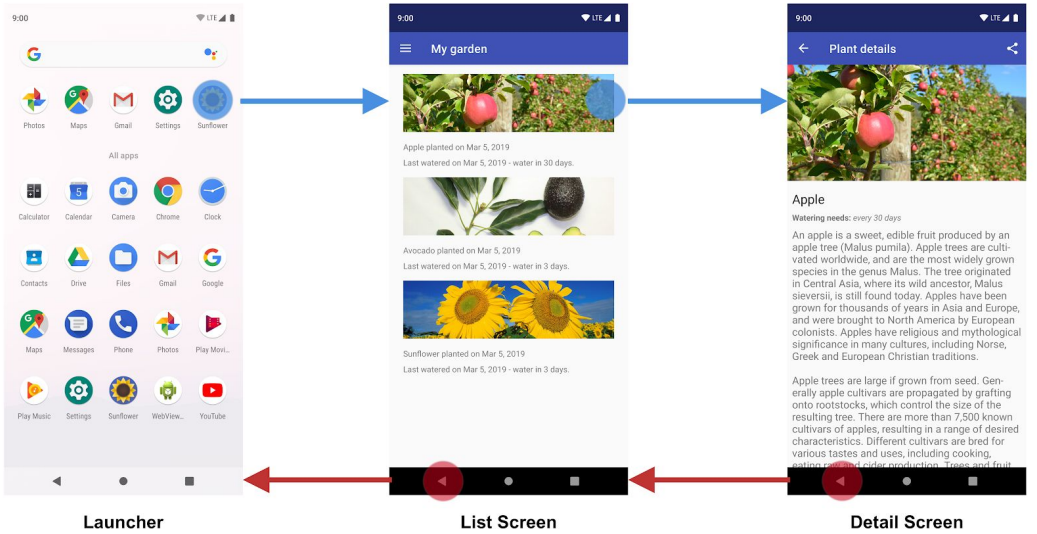


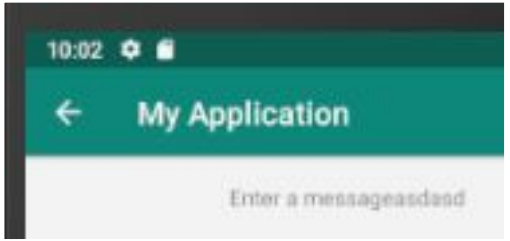
# App Architecture



# Navigation

- All apps have a fixed start location.
- A stack of Activities is maintained. When you press back, you pop from the stack.
- Also provide “up navigation”.
  - (exit a chain to a set location)
  - Define a parent activity in the manifest.



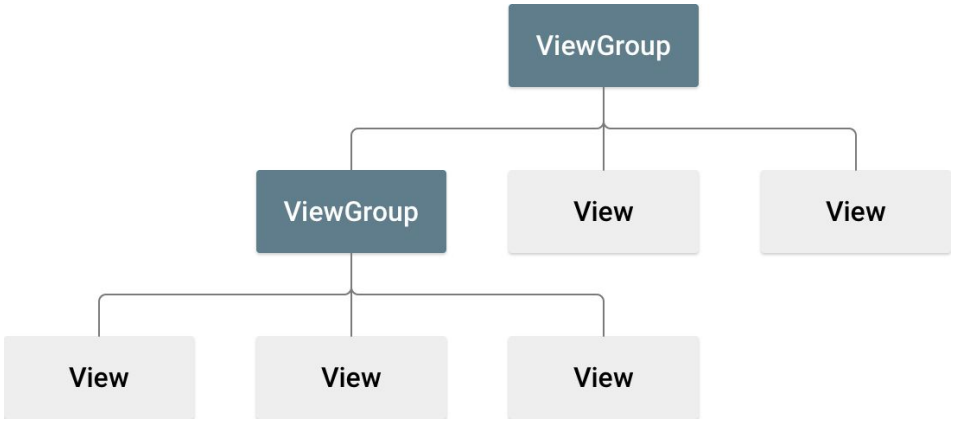
# Activity Best Practices

- Activities should coordinate with Data Models to retrieve a minimal amount of relevant data.
- Create independent, well-defined code modules.
- Make each module testable in isolation.
- Do not write code if an existing Activity does it.
- Use Models to persist fresh, relevant data.
- Assign one data source as the source of truth.

# Designing User Interfaces

# UI Design - Views and Layouts

- A layout (`ViewGroup`) defines the structure of the UI.
  - Containers that group one or more widgets (`View`).
    - A button, a text box.
- Many pre-defined types of layouts (`LinearLayout`, `Constraint Layout`).
- UI elements can be declared in XML or in code.



# UI Design - XML

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

Layout has two widgets, which have no constraints on each other's size or location

Text box containing a set string

Button with a set string.

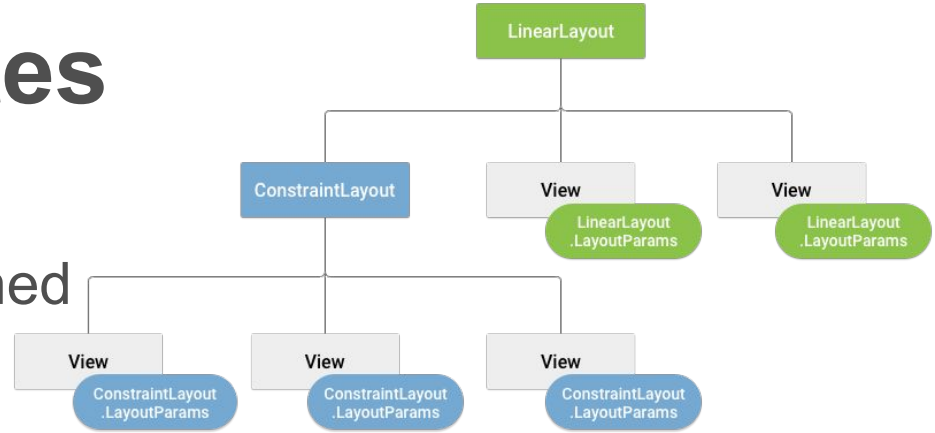
# UI Design - Attributes

- All View objects have a unique identifier.
  - Integer assigned at compile time, mapped to a user-specified variable: `android:id="@+id/my_button"`

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

```
Button myButton = (Button) findViewById(R.id.my_button);
```

# UI Design - Attributes



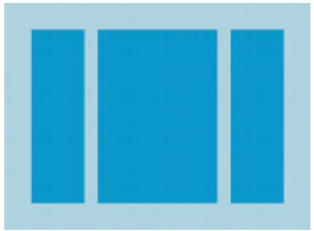

- LayoutParameters
  - Inherited by Views contained in that layout.
  - Define how Views appear.
- All ViewGroups have a width and height.
  - Each view must define width and height relative to this.
  - `wrap_content` sizes view to its content.
  - `match_parent` makes view as big as its parent ViewGroup allows.
  - Specify in dp (density-independent pixel units)

# UI Design - Layouts

- Views are rectangles with left and top coordinates.
  - Can get location with `getLeft()` and `getTop()`
  - Defined relative to the parent.
- Size is defined in width and height.
  - Measured width/height are how big the view *wants* to be.
  - Drawing width/height are the actual size of the view on screen, after layout constraints.
  - These can differ.

# UI Design - Common Layouts

### Linear Layout



A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

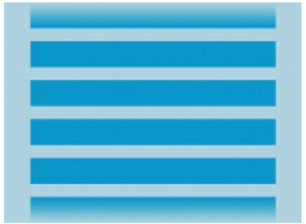
### Relative Layout

Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

### Web View

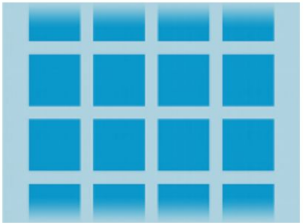

Displays web pages.

### List View



Displays a scrolling single column list.

### Grid View



Displays a scrolling grid of columns and rows.

Built from data using an Adapter

# UI Design - Responsive Design

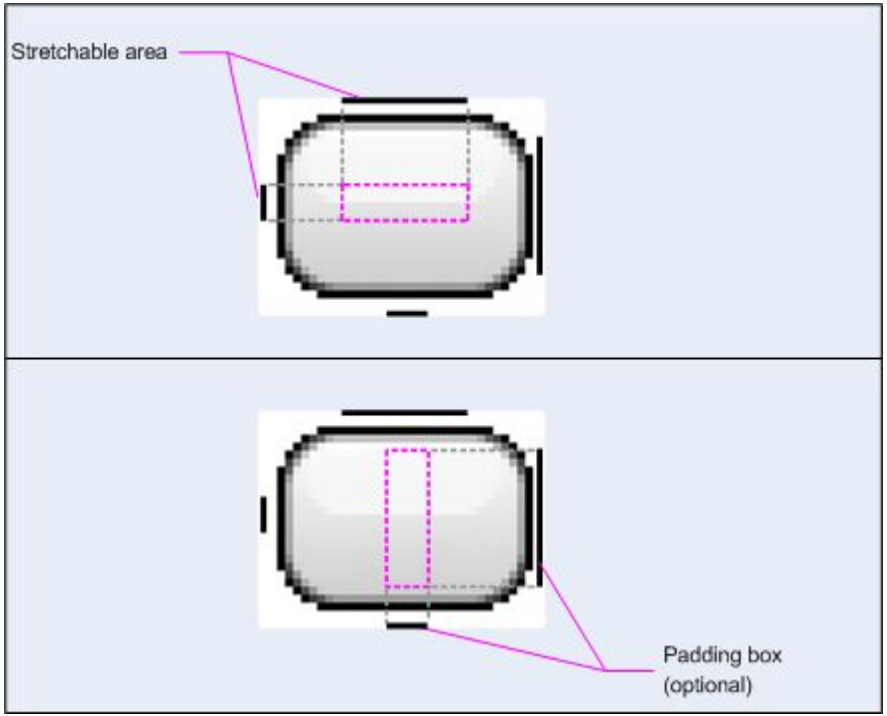
- Android defines two characteristics for each screen:
  - Screen Size (physical size)
    - Small, Normal, Large, XLarge
  - Screen Density (density of pixels on screen)
    - MDPI (~160dpi), HDPI (~240dpi), XHDPI (~320dpi), XXHDPI (~480dpi), XXXHDPI (~640dpi)
- Apps are compatible with all screen sizes and densities automatically, but this may not create a good UX.
  - Create specialized layouts, optimize images for density.

# Creating a Flexible Layout

- `ConstraintLayout` allows position/size specification based on spatial relationships between views.
  - All views move together as screen size changes.
  - Easiest to create in Android Studio Layout Editor.
- Avoid hard-coded layout sizes.
  - Use wrap_content, match_parent.
  - Automatically adjusts based on size and orientation of screen.

# Create Stretchable Images

- Bitmaps stretch to the screen size and density.
  - This can cause blurring and scaling artifacts.
- Nine-patch bitmaps add a 1px border that indicates which pixels can be stretched.
  - Intersection between left/top lines indicates the area that can be stretched.



# Demonstration - UI Design

Want to work along? This demonstration follows:
https://developer.android.com/training/basics/firstapp

You will need to install Android Studio, at least one emulator, and download the sample project.

This demonstration follows sections:

- Build a simple user interface
- Start another activity

# HTTP Requests - Volley Library

```
RequestQueue queue = Volley.newRequestQueue(this);
String url ="http://www.google.com";

// Request a string response from the provided URL.
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
            new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {
        // Display the first 500 characters of the response string.
        mTextView.setText("Response is: "+ response.substring(0,500));
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        mTextView.setText("That didn't work!");
    }
});

// Add the request to the RequestQueue.
queue.add(stringRequest);
```

- Create a `RequestQueue` and pass `Request` objects.
- Thread safe
- Requires INTERNET permission in manifest.

https://developer.android.com/training/volley/simple#java

# What's Next?

- Wednesday: Android Supervision
- Thursday: More Android!
  - Testing
  - Profiling
  - Processes and Threads
  - Services
  - Broadcast Receivers
  - Content Providers


UNIVERSITY OF
GOTHENBURG


CHALMERS
UNIVERSITY OF TECHNOLOGY