COMP2000: Software Engineering 2

Lecture-4: Introduction to Android Development

Dr Vivek Singh

Lecturer in Artificial Intelligence

University of Plymouth

UNIVERSITY OF PLYMOUTH

Outline

- Overview of mobile apps
- Getting started
- Application Fundamentals
- App components
- User interfaces

UNIVERSITY OF PLYMOUTH

Mobile apps

- by definition, are software applications that run on portable devices such as smartphones and tablets.
- These apps are available for download through device-specific portals like Google Play Store and are installed onto users' devices.
- There are three types of mobile applications you can pick from:
- Native (this type will be considered in this module)
- Web
- Hybrid

Native Apps

- Definition: Built specifically for one operating system, utilizing the platform’s official development language.
- Programming Language: Android apps are developed in Java or Kotlin, while iOS apps are developed in Swift or Objective-C.
- Advantages: Full access to device features, high performance, and a smooth user experience.
- Example: Instagram, developed as a native app for Android and iOS.

Web Apps

- Definition: Mobile-optimized websites that look and function similarly to native apps but run in a web browser.
- Technologies Used: Primarily HTML5, CSS, and JavaScript.
- Limitations: Limited access to device features and typically requires an internet connection.
- Example: Google Maps (when accessed via a mobile browser).

Hybrid Apps

- Definition: A blend of native and web apps, utilizing a web view within a native shell.
- Technologies Used: Built using HTML5, CSS, and JavaScript, but wrapped in a native container to allow installation from app stores.
- Advantages: Cross-platform code reuse with some access to device features.
- Example: Netflix, which uses a hybrid structure to run on both desktop and mobile.

UNIVERSITY OF PLYMOUTH

Cross-Platform Frameworks

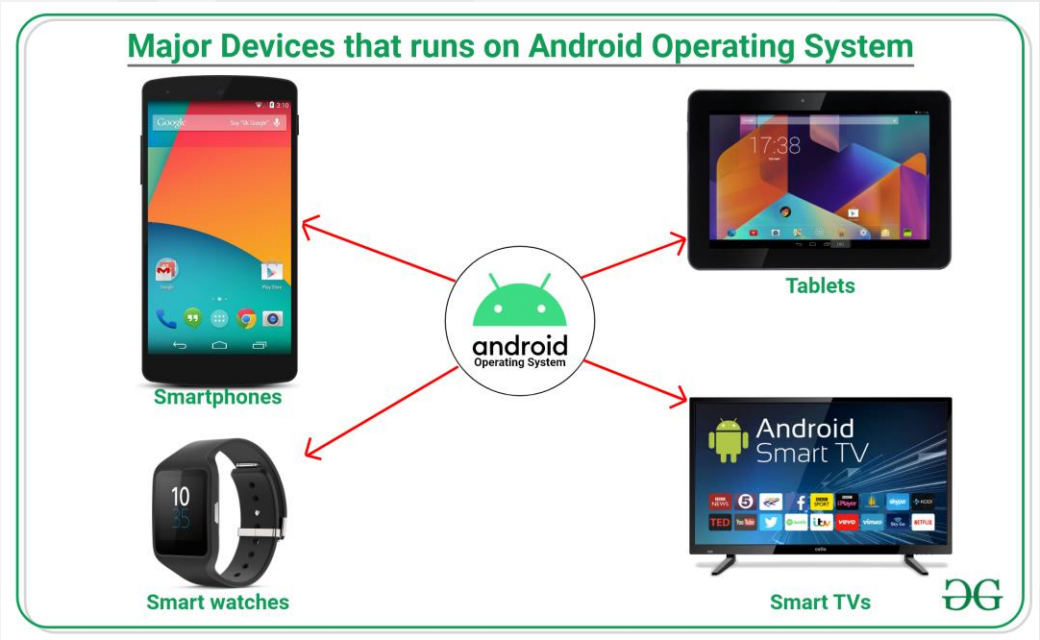
- Definition: Frameworks that allow developers to build applications for multiple platforms using a single codebase.
- Technologies Used: Popular frameworks include Flutter (Dart) and React Native (JavaScript).
- Advantages: Reduces development time and cost by enabling code sharing across platforms, while still allowing access to most native features.
- Example: Apps like Alibaba and Google Ads, built with Flutter, or Facebook, which leverages React Native.

Native Android apps

- Apps written specifically for the Android Platform.
- Main focus of this module.

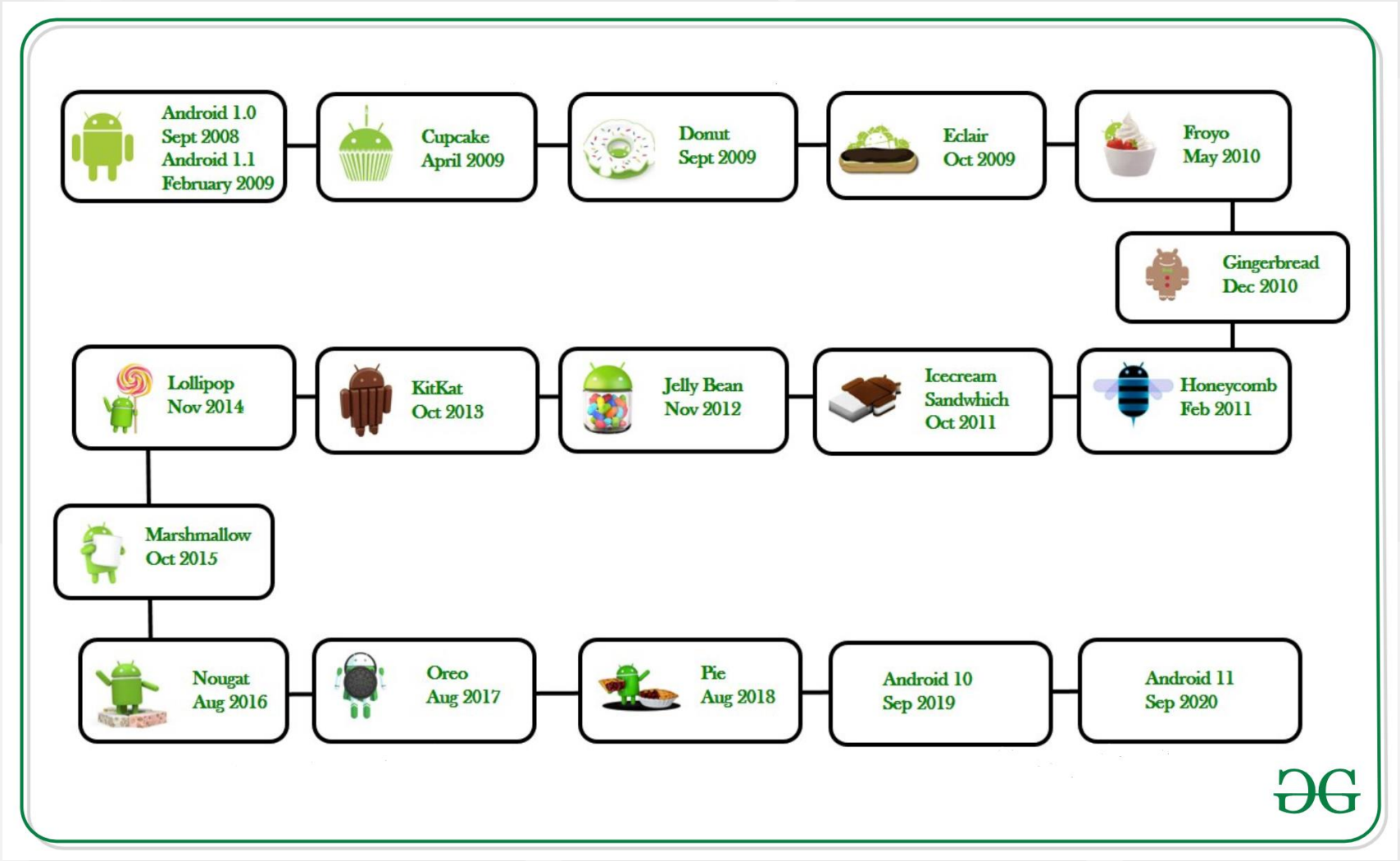
UNIVERSITY OF PLYMOUTH

Android operating system



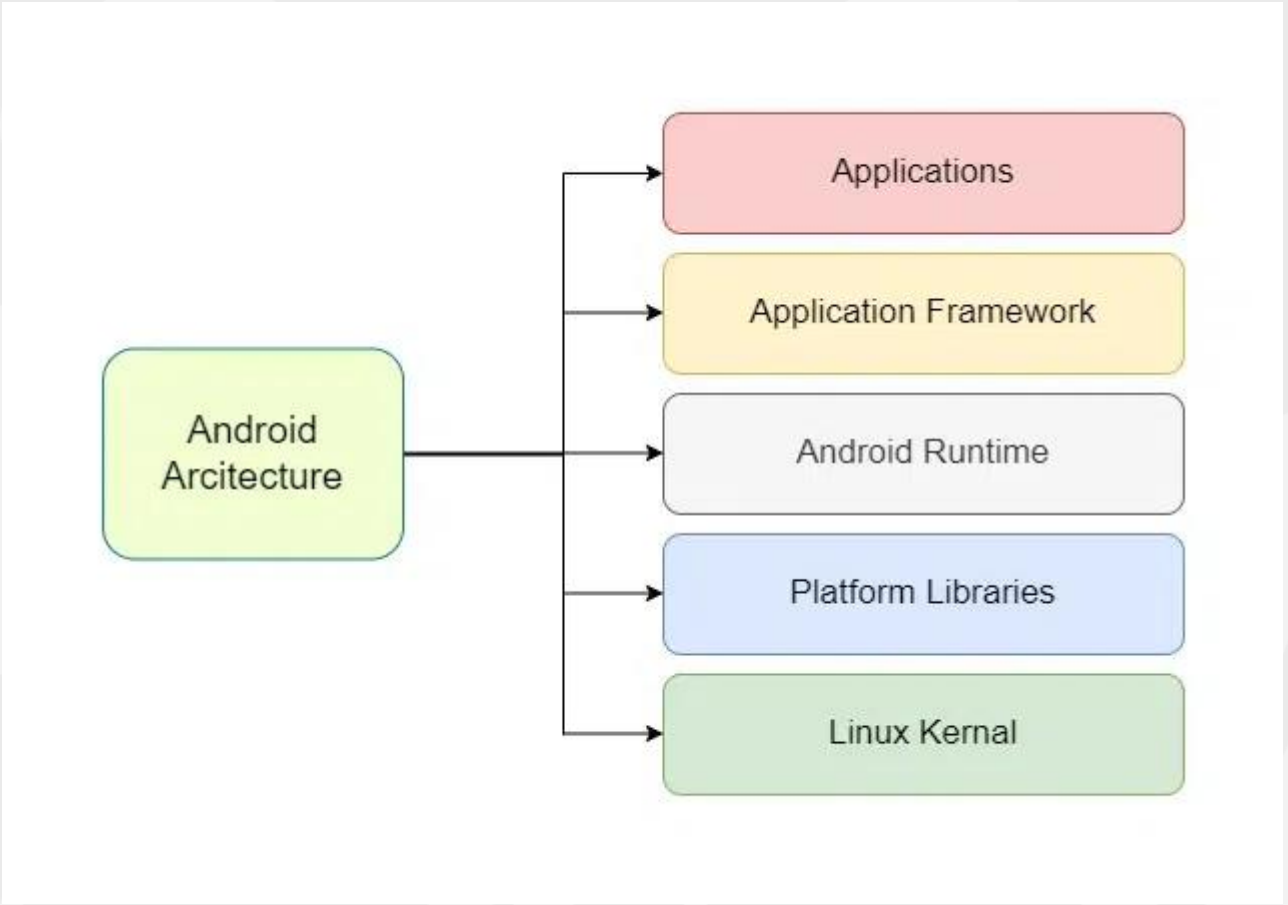
- largest installed base among various mobile platforms across the globe.
- Conquered 71% of the global market share by the end of 2021.
- September 2008, the first Android-powered device was launched.
- code libraries and its popularity has made Android OS of choice for softwares for all devices like tablets, wearables, set-top boxes, smart TVs, notebooks, etc.

UNIVERSITY OF PLYMOUTH


Android 1.0
Sept 2008
Android 1.1
February 2009
Cupcake
April 2009
Donut
Sept 2009
Eclair
Oct 2009
Froyo
May 2010
Gingerbread
Dec 2010
Honeycomb
Feb 2011
Icecream
Sandwhich
Oct 2011
Jelly Bean
Nov 2012
KitKat
Oct 2013
Lollipop
Nov 2014
Marshmallow
Oct 2015
Nougat
Aug 2016
Oreo
Aug 2017
Pie
Aug 2018
Android 10
Sep 2019
Android 11
Sep 2020

Key Components of Android

- Applications Layer
- Application Framework
- Android Runtime (ART)
- Platform Libraries
- Linux Kernel



Applications Layer

- This is the top layer, where user-installed applications, like Contacts, Messages, and Games, reside.
- These are the apps that users interact with directly.
- Acts as the interface for end users, enabling them to interact with the device and perform tasks using various applications.

Application Framework

- This layer provides a toolkit for developers, offering a set of APIs that applications can utilize.
- enables app developers to build complex applications by reusing core components
- Key Components:
  - Activity Manager: Manages the lifecycle of applications and activities.
  - Content Providers: Manages app data sharing.
  - Resource Manager: Manages resources like graphics and UI layout files.
  - View System: Provides the UI elements like buttons, text fields, etc.

Android Runtime (ART)

- The Android Runtime (ART) is responsible for executing Android applications.
- It replaces the older Dalvik Virtual Machine for enhanced performance.
- Key Components:
  - Core Libraries: Provides the essential classes and functionalities for Java development.
  - Ahead-of-Time (AOT) Compilation: Improves app performance by compiling bytecode to machine code upon installation.

Platform Libraries

- This layer consists of C/C++ libraries that support various system features,
  - such as graphics rendering, database access, and web browsing.
- Key Libraries:
  - SQLite: Manages database access and is often used for local storage.
  - OpenGL ES: Handles graphics rendering for 2D and 3D graphics.
  - WebKit: Enables web browsing capabilities within applications.

Linux Kernel

- The foundation of the Android operating system, based on the Linux kernel
- provides low-level functionalities and hardware abstraction.
- Key Components:
  - Memory Management: Efficient handling of memory resources.
  - Process Management: Manages processes and enforces security between applications.
  - Networking: Supports networking protocols for internet connectivity.
  - Device Drivers: Manages hardware components, such as the camera, Bluetooth, and Wi-Fi.

UNIVERSITY OF PLYMOUTH

Getting Started

Download Android SDK
Download the latest version of Android studio:

https://developer.android.com/studio

IntelliJ and Android studio are installed in the **web development lab**

UNIVERSITY OF PLYMOUTH

Application Fundamentals

- Android apps can be written using Kotlin and Java languages.
- The Android SDK tools compile your code along with any data and resource files into an APK or an Android App Bundle.
- Android SDK includes the compiler, debugger, and a device emulator.

Application Fundamentals

- Sandboxing:
  - Each Android application runs in a sandbox environment, which isolates it from other apps.
  - This ensures that one app cannot access another app's data or interfere with its functioning.
- Android's underlying Linux kernel treats each app as a separate user.
- This is implemented by assigning a unique Linux user ID to each app.

UNIVERSITY OF PLYMOUTH

- Every Android app runs in its own virtual machine (ART), allowing its code to execute independently.
- This isolation enhances security and stability, as one app crashing does not affect others.
- By default, each app operates within its own Linux process, which is managed by the system.
- The process is created when any of the app's components need to run and is terminated when no longer required, optimizing memory usage.

- **App Lifecycle Management**:
- Android actively manages app processes
- When an app component (like an Activity or Service) needs to run, the Android system starts the process.
- When the system needs to reclaim resources, it may terminate processes for apps that are not in active use.

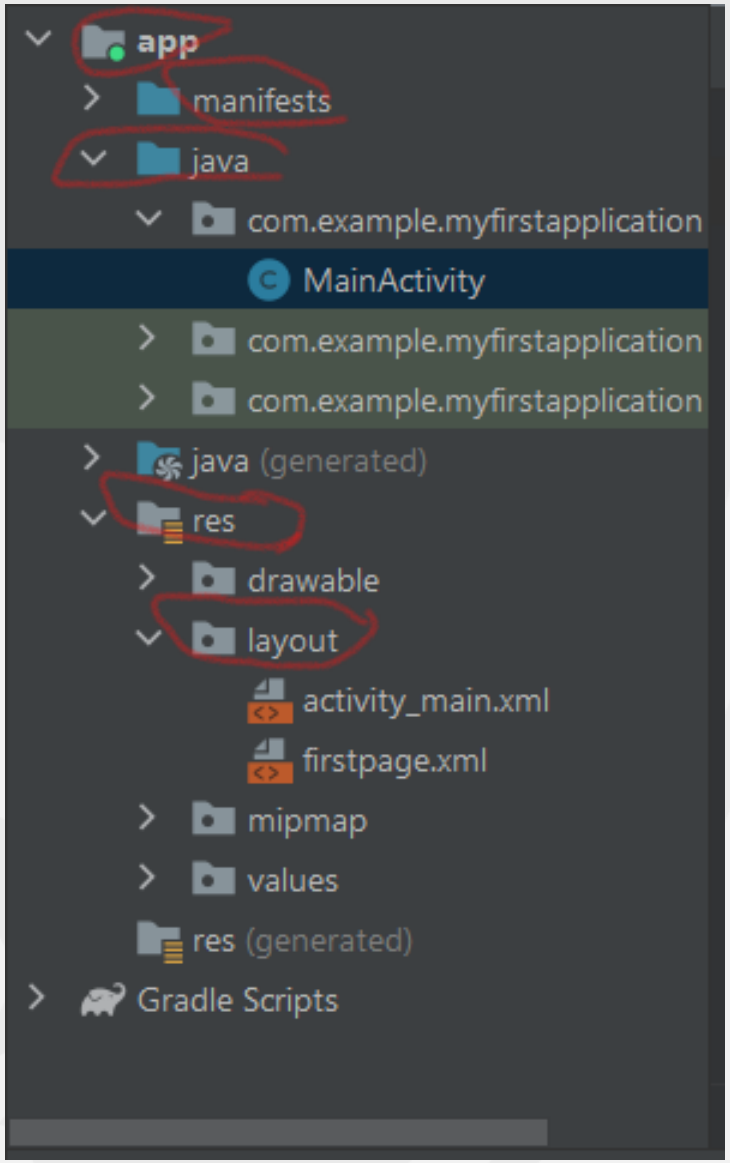
Important files

app > java > com.example.myfirstapp > MainActivity

This is the main activity. It's the entry point for your app. When you build and run your app, the system launches an instance of this Activity and loads its layout.

Contains the core logic and code that defines the behavior of the main screen.

Activities are a fundamental part of any Android app, acting as the bridge between the user interface (UI) and the application's functionality.



Important files

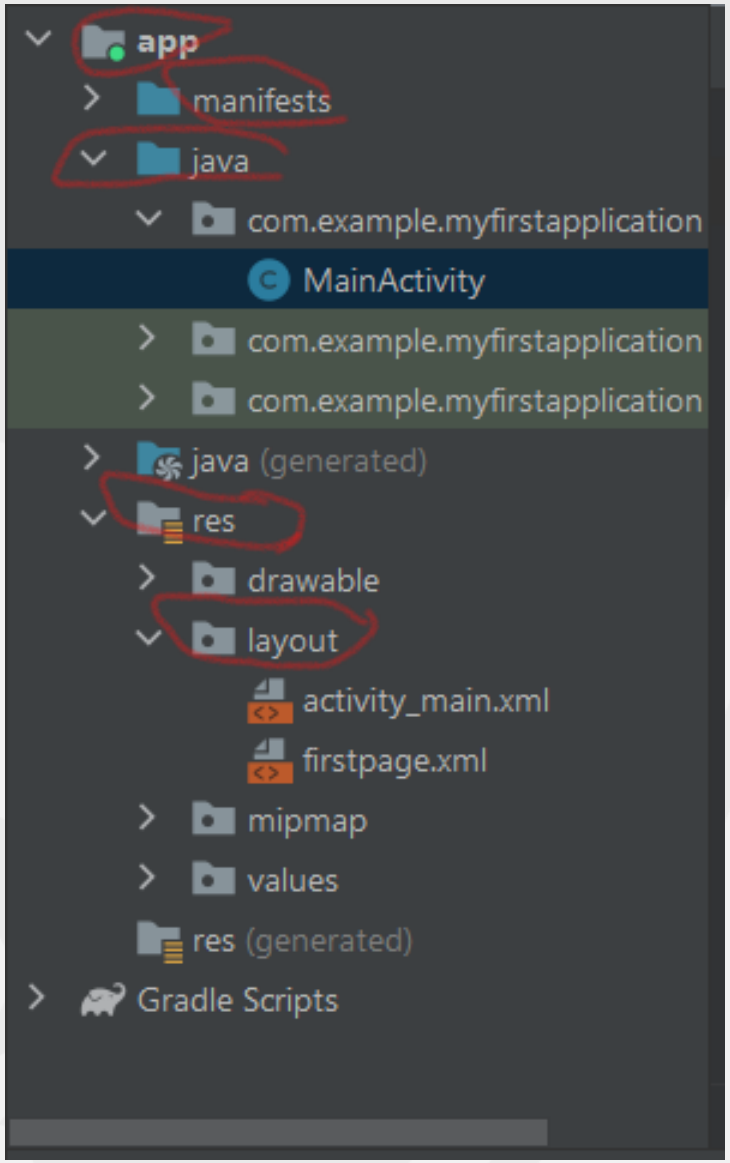
app > res > layout > activity_main.xml

This XML file defines the layout for the activity's user interface (UI).

It contains a TextView element with the text "Hello, World!“ as default value.

It specifies the arrangement and properties of the UI components, such as buttons, text fields, and images.

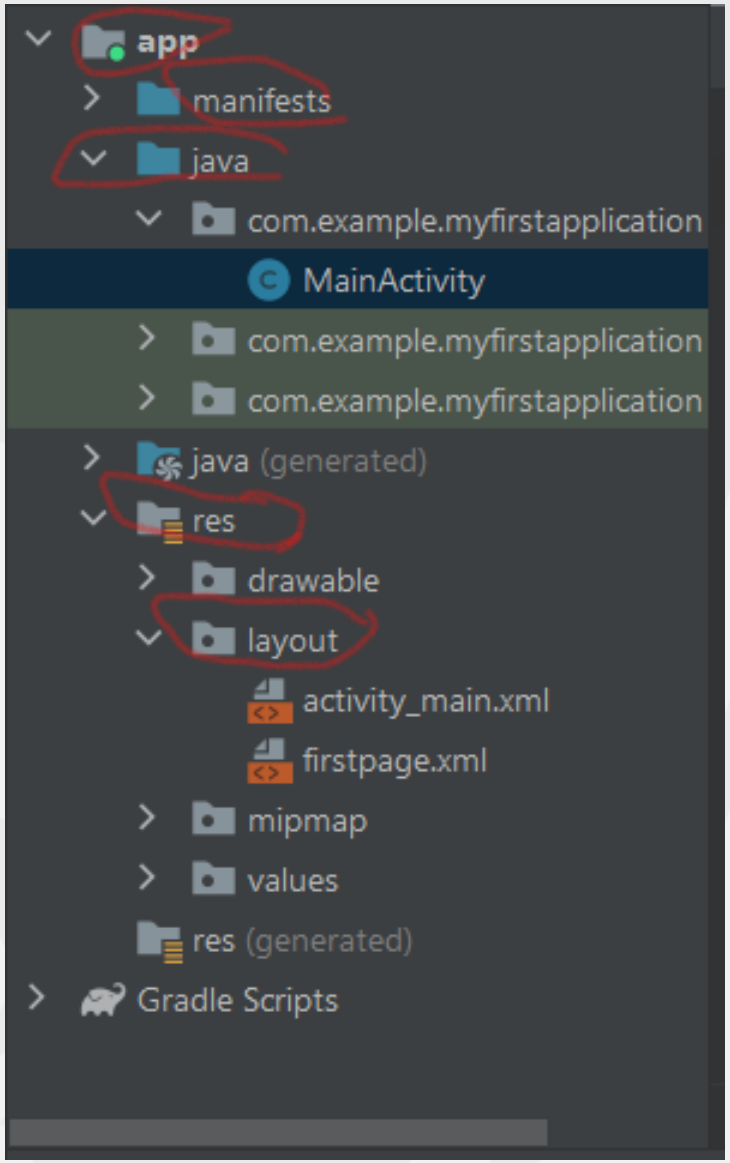
By separating layout definitions into XML files, Android allows for a clear separation of the user interface from application logic.



Important files

app > manifests > AndroidManifest.xml

- The manifest file describes the fundamental characteristics of the app and defines each of its components.
- Declares essential components like Activities, Services, Broadcast Receivers, and Content Providers.
- Specifies permissions the app requires, such as Internet access or GPS usage.
- Sets the application's metadata, including its name, icon, and theme.
- Without this file, the Android system cannot recognize the app’s structure and permissions, making it impossible for the app to run properly.



UNIVERSITY OF PLYMOUTH

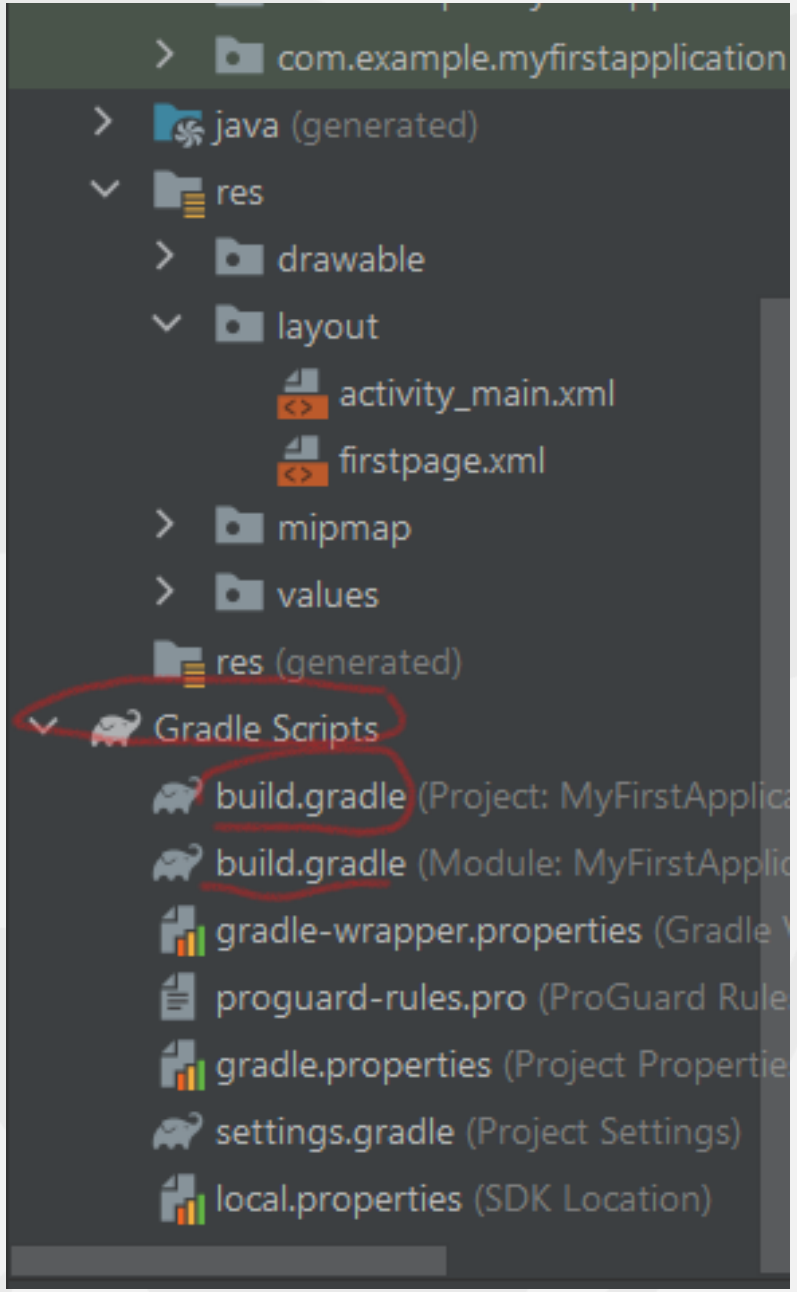
Gradle Scripts > build.gradle

Gradle is a build automation tool used to manage dependencies, configure build settings, and automate tasks such as packaging the app.

There are two files with this name: one for the project, "Project: My_First_App," and one for the app module, "Module: My_First_App.app.“

Each module has its own build.gradle file, but this project currently has just one module.

Use each module's build.gradle file to control how the Gradle plugin builds your app.



UNIVERSITY OF PLYMOUTH

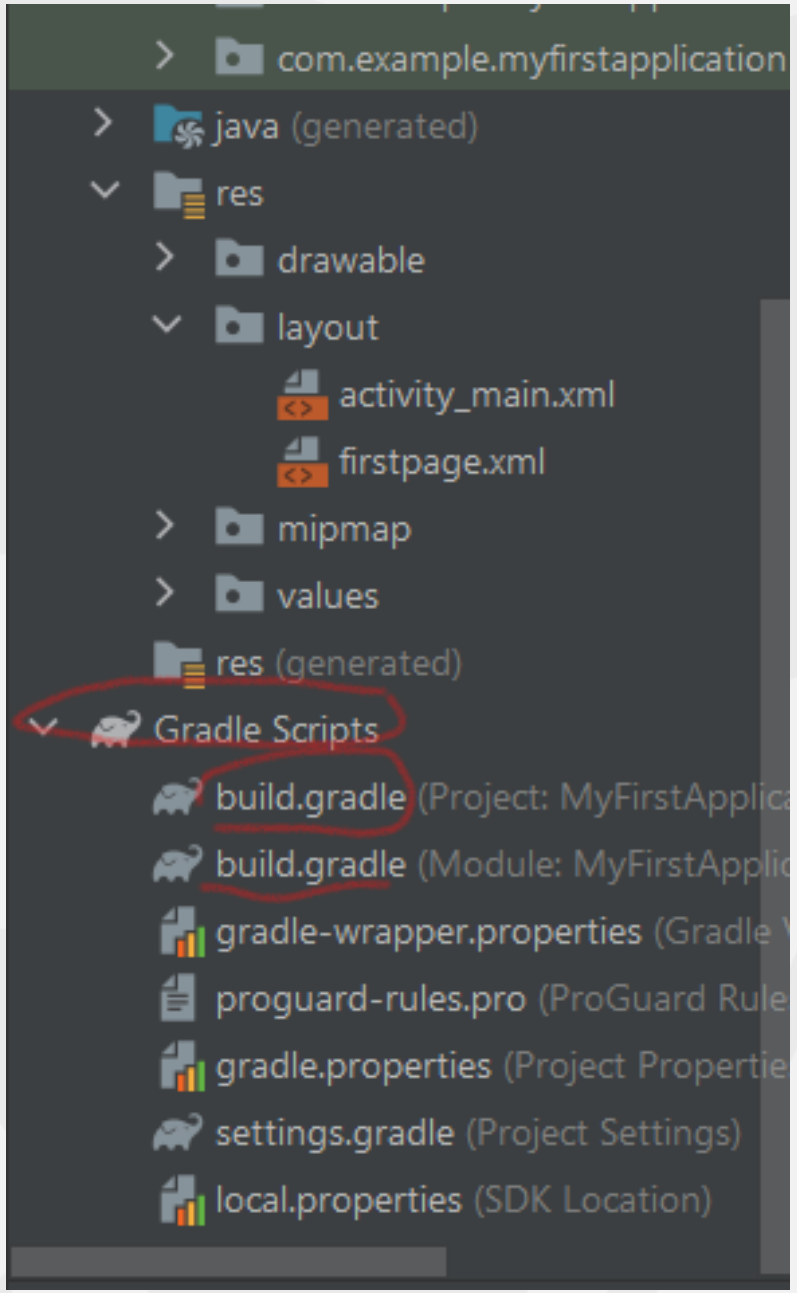
Gradle Scripts > build.gradle

Project-Level gradle File

- Defines the Gradle plugin versions used by the project.
- Specifies repositories (e.g., Google's Maven repository) where dependencies are fetched from.
- Configures build settings that apply across all modules in the project.

Module-Level gradle File

- Defines dependencies for particular module, such as Android libraries, third-party libraries, or other modules.
- Specifies the application's compile SDK version, min SDK version, and target SDK version, which define the Android API levels the app is compatible with.
- Configures build types (e.g., debug and release builds) for creating different versions of the app.



App components

- App components are the essential building blocks of an Android app.
- There are four different types of app components:

- Activities
- Services
- Broadcast receivers
- Content providers

App components

- Activity: An activity represents a single, focused screen with a user interface, such as a login screen, main menu, or settings page.
  - It is the entry point for interacting with the user.
  - Each screen in an Android app typically has its own activity.
  - For example, in a messaging app, there might be one activity for viewing messages and another for composing a new message.

- Service: A service is a background component that performs long-running tasks without requiring a user interface.
  - Services handle tasks that need to continue working even when the user is not actively interacting with the app
  - such as music playback, file downloads, or data syncing.
  - Services ensure that these processes are maintained independently from user activities.

- Broadcast Receiver: A broadcast receiver allows the app to listen for and respond to system-wide broadcast announcements, even when the app is not running.
  - Broadcast receivers can respond to events such as battery changes, network connectivity changes, or incoming messages.
  - They enable apps to be notified of system events and take appropriate action, such as displaying a notification or updating data.

- Content Provider: A content provider manages access to a structured set of data, often shared across multiple applications.
  - Content providers handle data storage, allowing other apps to query or modify data.
  - They can store data in various locations, such as a SQLite database or cloud storage, and enable secure sharing of app data.
  - For example, the Contacts app uses a content provider to allow other apps to access contact information.

- **Activity**, **Service**, and **Broadcast Receiver**—are activated by **Intents**.

- An Intent is an asynchronous message that provides a way to bind different components to each other at runtime
- Intents can start new activities, initiate services, and deliver broadcasts

The manifest file

- Before the Android system can start an app component, the system must know that the component exists by reading the app's *manifest file*, *AndroidManifest.xml*.
- Your app must declare all its components in this file, which must be at the root of the app project directory.

The manifest does a number of things in addition to declaring the app's components, such as the following:

- Identifies any user permissions the app requires, such as Internet access or read-access to the user's contacts.
- Declares the minimum API Level required by the app, based on which APIs the app uses.
- Declares hardware and software features used or required by the app, such as a camera, Bluetooth services, or a multitouch screen.
- Declares API libraries the app needs to be linked against (other than the Android framework APIs), such as the Google Maps library.

How to Declare components?

- The primary task of the manifest is to inform the system about the app's components. For example, a manifest file can declare an activity as follows:

```
<?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>
```

In the <application> element, the android:icon attribute points to resources for an icon that identifies the app.

In the <activity> element, the android:name attribute specifies the fully qualified class name of the Activity subclass and the android:label attribute specifies a string to use as the user-visible label for the activity.

You must declare all app components using the following elements:

- <activity> elements for activities.
- <service> elements for services.
- <receiver> elements for broadcast receivers.
- <provider> elements for content providers.

Examples of manifest components

The Android Security model allows an app to do pretty much anything it likes.

But you must ask for permission to do with pretty much everything the app wants to do in its manifest., e.g. to make network connections,

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

Examples of manifest components

- minimum API level that the app supported
  - <uses-sdk android:minSdkVersion="21" />
- any hardware or software features that the app needs
  - <uses-feature android:name="android.hardware.camera" />

Android User Interface Fundamentals

- All visual components in Android descend from the View class and are referred to generically as *Views*. You'll often see Views referred to as *controls* or *widgets*— terms you're probably familiar with if you've previously done any GUI development.

- The ViewGroup class is an extension of View designed to contain multiple Views. View Groups are used most commonly to manage the layout of child Views, but they can also be used to create atomic reusable components.

- View Groups that perform the former function are generally referred to as *layouts*.

UNIVERSITY OF PLYMOUTH

- The user interface (UI) for an Android app is built as a hierarchy of *layouts* and *widgets*.

- The layouts are ViewGroup objects, containers that control how their child views are positioned on the screen. Widgets are View objects, UI components such as buttons and text boxes.

Layout

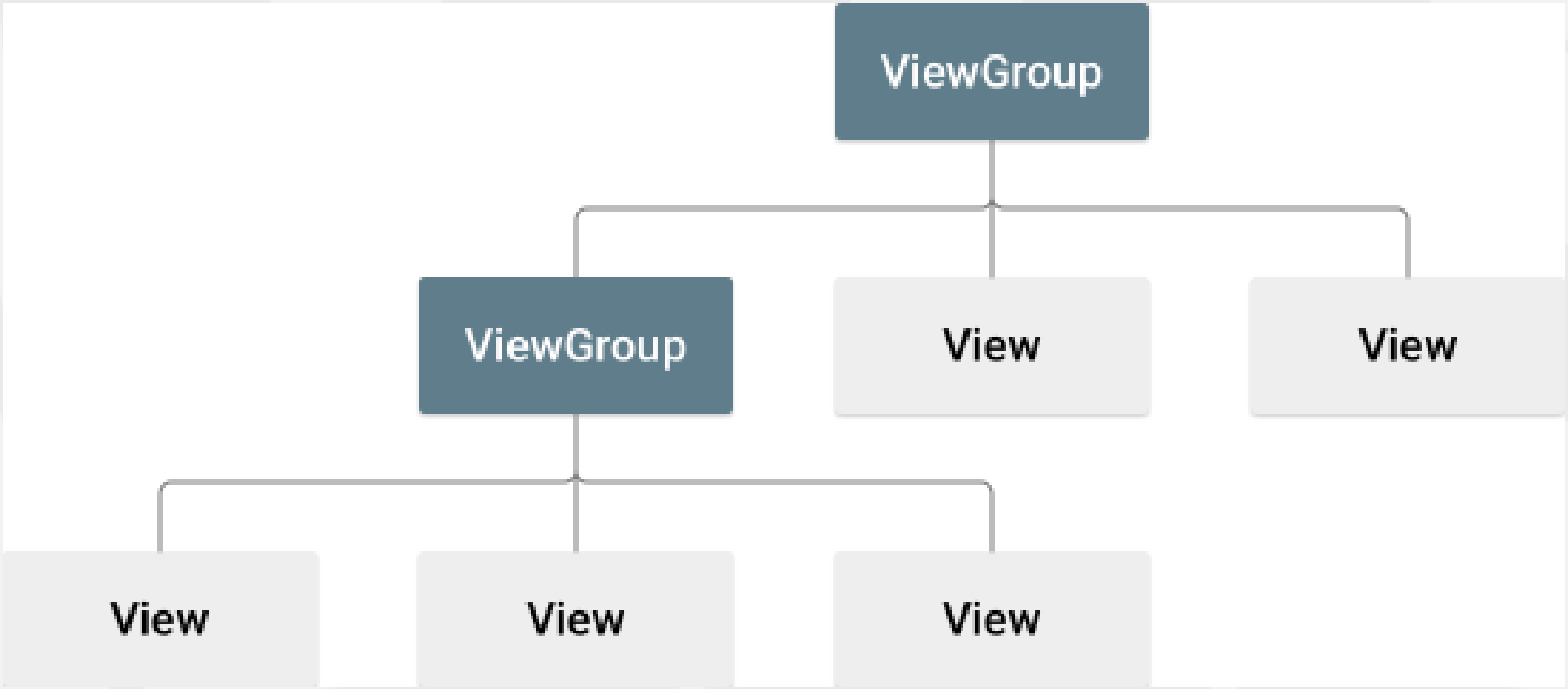


Figure 1: Illustration of a view hierarchy, which defines a UI layout (Android developer website)

User interface

http://eagle.phys.utk.edu/guidry/android/androidUserInterface.html

- Views: Basic building blocks of the UI. All UI controls, including widgets like TextView, Button, and ImageView, are derived from the View class.
  - Purpose: Display visual elements or handle user interaction.
  - Examples: Controls (widgets) such as buttons, labels, and images

- ViewGroups: Containers for Views and other ViewGroups, used to define the layout structure.
  - Purpose: Organize and manage the layout of child Views. They provide Layout Managers to arrange Views.
  - Examples: LinearLayout for vertical or horizontal arrangement, RelativeLayout for positioning Views relative to each other, and ConstraintLayout for complex UI designs.

- Fragments: Encapsulated UI components within an Activity. They are reusable and adaptable, especially useful for optimizing UIs on different screen sizes.
  - Purpose: Enable modularity and reusability of the UI. Multiple fragments can be combined in a single Activity, especially useful for tablets or large screens.
  - Example: A Fragment displaying a list of items on the left and a detailed view on the right for larger screens.
- Activities: Containers for the app's UI, acting as the entry points for user interaction. They control the lifecycle of the UI.
  - Purpose: Display Views and Fragments. Each Activity represents a screen in the app and handles user navigation.
  - Example: A messaging app with separate Activities for the main conversation list and the chat window.

- **Views** — Views are the base class for all visual interface elements (commonly known as *controls* or *widgets*). All UI controls, including the layout classes, are derived from View.

- **View Groups** — View Groups are extensions of the View class that can contain multiple child Views. Extend the ViewGroup class to create compound controls made up of interconnected child Views.

- The **ViewGroup** class is also extended to provide the Layout Managers that help you lay out controls within your Activities.

- **Fragments** — Fragments, are used to encapsulate portions of your UI.
- This encapsulation makes Fragments particularly useful when optimizing your UI layouts for different screen sizes and creating reusable UI elements.
-
- **Activities** — Activities represent the window or screen, being displayed (will be described in next lecture).
- Activities are the Android equivalent of Forms in traditional Windows desktop development.
- To display a UI, you assign a View (usually a layout or Fragment) to an Activity.

Introducing Layouts

- **Layout Managers**: Layout Managers, or simply **layouts**, are extensions of the **ViewGroup** class. They control the positioning of child Views within the UI.
- Layouts can be **nested** to create complex UI arrangements by combining multiple layout types.
- The Android SDK offers a variety of layout classes that you can use directly, or modify to design the UI for your Views, Fragments, and Activities.
- Selecting the right layout combination is crucial for creating an aesthetically pleasing, user-friendly, and performance-efficient UI.

UNIVERSITY OF PLYMOUTH

Layout Types and Their Functions

- FrameLayout — The simplest of the Layout Managers, the Frame Layout pins each child view within its frame.
  - The default position is the top-left corner, though you can use the *gravity* attribute to alter its location.
  - Adding multiple children stacks each new child on top of the one before, with each new View potentially obscuring the previous ones.

- LinearLayout — A Linear Layout aligns each child View in either a vertical or a horizontal line.
  - A vertical layout has a column of Views, whereas a horizontal layout has a row of Views.
  - The Linear Layout supports a *weight* attribute for each child View that can control the relative size of each child View within the available space.

Layout Types and Their Functions

- TableLayout: Arranges child Views into rows and columns, similar to an HTML table.
  - Each row within a TableLayout is a TableRow ViewGroup, allowing you to organize elements in a structured grid.
  - Useful for forms or settings screens where you want to align items in a grid-like fashion.
- ConstraintLayout: as a more powerful and flexible layout manager, it allows you to create complex layouts by defining constraints between Views.
  - Ideal for building responsive and scalable UIs, especially in cases where you want to avoid deep layout nesting.

Design Considerations for Layouts

- Each layout class is designed to scale to different screen sizes, avoiding the use of absolute positions or fixed pixel values.
- Use layout properties like wrap_content and match_parent to create adaptable UIs.
- Avoid deep nesting to optimize performance and reduce layout rendering time.

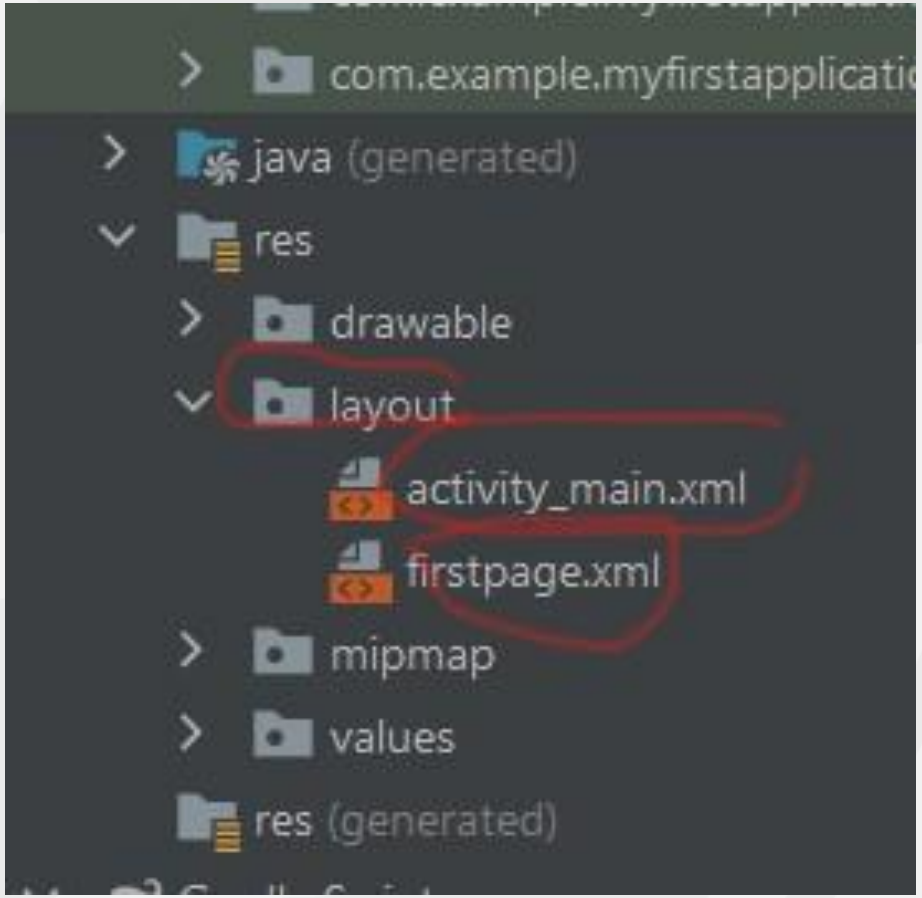
Layout resources

A layout resource defines the architecture for the UI in an Activity or a component of a UI.

File location:
res/layout/*filename*.xml
The filename will be used as the resource ID.

Compiled resource datatype:
Resource pointer to a View (or subclass) resource.

Resource reference:
In Java: R.layout.*filename*
In XML: @[*package*:]layout/*filename*



Defining Layouts

- The preferred way to define a layout is by using XML external resources.
- Each layout XML must contain a single root element. This root node can contain as many nested layouts and Views as necessary to construct an arbitrarily complex UI.
- The following snippet shows a simple layout that places a *TextView* above an *EditText* control using a vertical *LinearLayout*.

Syntax

```
<?xml version="1.0" encoding="utf-8"?>
<ViewGroup
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@[+][package:]id/resource_name"
    android:layout_height=["dimension" | "match_parent" | "wrap_content"]
    android:layout_width=["dimension" | "match_parent" | "wrap_content"]
    [ViewGroup-specific attributes] >
    <View
        android:id="@[+][package:]id/resource_name"
        android:layout_height=["dimension" | "match_parent" | "wrap_content"]
        android:layout_width=["dimension" | "match_parent" | "wrap_content"]
        [View-specific attributes] >
        <requestFocus/>
    </View>
    <ViewGroup >
        <View />
    </ViewGroup>
    <include layout="@layout/layout_resource"/>
</ViewGroup>
```

Note: The root element can be either a **ViewGroup**, a **View**, or a **<merge>** element, but there must be only one root element and it must contain the **xmlns:android** attribute with the **android** namespace as shown.

XML:

```
<TextView android:id="@+id/nameTextbox"/>
```

Java:

```
TextView textView = findViewById(R.id.nameTextbox);
```

```
<?xml version=”1.0” encoding=”utf-8”?>
<LinearLayout
    xmlns:android=”http://schemas.android.com/apk/res/android”
    android:orientation=”vertical”
    android:layout_width=”match_parent”
    android:layout_height=”match_parent”>
<TextView
      android:layout_width=”match_parent”
      android:layout_height=”wrap_content”
      android:text=”Enter Text Below”
    />
<EditText
       android:layout_width=”match_parent”
       android:layout_height=”wrap_content”
       android:text=”Text Goes Here!”
    />
</LinearLayout>
```

- When preferred, or required, you can implement layout components in code.
- When assigning Views to layouts in code, it's important to apply LayoutParameters using the setLayoutParams method, or by passing them in to the addView call:

Start your first Project

- To create your new Android project, follow these steps:
- Install the latest version of Android Studio.
- In the **Welcome to Android Studio** window, click **Start a new Android project**.
- If you have a project already opened, select **File** > **New** > **New Project**.
- In the **Choose your project** window, select **Empty Activity** and click **Next**.
- In the **Configure your project** window, complete the following:
  - Enter "My First App" in the **Name** field.
  - Enter "com.example.myfirstapp" in the **Package name** field.
  - If you'd like to place the project in a different folder, change its Save location.
  - Select either **Java** from the Language drop-down menu.
  - **Leave the other options as they are.**
- Click **Finish**

UNIVERSITY OF PLYMOUTH

Resources

- Android developer website: https://developer.android.com/docs

Thank you

UNIVERSITY OF PLYMOUTH