# Android Application Structure

# Android Application Structure

- For your research focus on the following
  - The structure of an actual android project
  - The logical structure of a running app (intents etc) and the reasoning behind it

# Android Application Structure

- In this set of lecture notes we will explore the structure of an android application
  - Before it gets turned into a single APK file that gets installed on the user's device
  - There are a lot of support files that an android application requires in order to function
  - We will explore the motivations for these decisions to split out the application into several components

# Code vs Non-Code resources

- The first thing you will notice when you look at the structure of an android application is that it is split into two main components
  - Code resources and non code resources
  - Code resources comprise all the java classes that you write for android (all your executable code)
  - Non-code resources comprise XML files, images and other static items

# Code vs Non-Code resources

- The original motivation for this arrangement is to take advantage of compression algorithms
  - Most non code resources are text format meaning that they will compress very well
  - Initially android devices came with very little space so reducing package size was a must
  - Thus making as many resources in texual format saved a lot of space when compressed

# Code vs Non Code resources

- Dalvik and ART will treat non-code resources in the same way
  - Will treat them like static files
- However code resources are treated differently
  - Dalvik will run them through a java interpreter while ART will compile them as they are installed (we will see why in the following weeks)

# Non-Code resources

- While the code resources are relatively simple in nature
  - Just comprising java code.
- Non code resources are much more complex
  - Many different files and directories serving different purposes.

# The res/ Directory

- All non code resources are held in a single directory
  - The res/ directory
- However it is comprised of many files and subfolders that each serve a different purpose
  - We will explore those in detail here

# The mipmap Directories

- When you construct an android application you may notice that their are multiple of these mipmap directories
  - Each with its own DPI level attached
- Each one of these directories contains a full copy of raster images designed to render on a device that supports that DPI level
  - This is to reduce the need for expensive image resizing operations

# The mipmap Directories

- Low DPI devices are likely to have very little power
  - Thus scaling large images down to size will result in a blocky low detail image
  - While also consuming much CPU on each resize
  - Making the application less responsive

# The mipmap Directories

- High DPI devices will have a lot more CPU power
  - Thus while scaling small images up to size will be quicker
  - We end up with a blurry image with little to no detail
- The solution to both problems is to maintain copies of all images for different DPI devices.

# The mipmap Directories

- It is important to note the definition of DPI
  - Dots Per Inch: is the number of pixels or dots that appear within 1 inch of physical screen space
  - Thus 160 DPI (android's reference screen size) states that there will be 160 pixels in one inch of screen space
  - The higher the DPI the more detailed the display

# The mipmap Directories

- There are six default levels that an android application can support
  - Drawable-ldpi (deprecated in API 14 onwards) covers devices of approximately 120 DPI
  - Drawable-mdpi covers devices of approximately 160 DPI
  - Drawable-hdpi covers devices of approximately 240 DPI

# The mipmap Directories

- Drawable-xhdpi covers devices of 320 DPI
- Drawable-xxhdpi covers devices of 480 DPI
- Drawable-xxxhdpi covers devices of 640 DPI

- Android devices depending on their screen size and resolution will class themselves into one of these 6 categories.
  - They will take the resulting images from that directory for displaying on screen

# Android Manifest

- Arguably the most important file in an android application
  - Responsible for telling an android device about the application that is being installed
  - States details like name and api required to run
  - Also states what permissions the application needs to run, so users know what an application will do.

# Android Manifest

- One of the most important functions of this file is to list the permissions need by an application
  - List of permissions will be shown to user before application is installed
  - Must be accepted by user before installation
    - This changed in 6.0
  - Apps that request too many permissions are unlikely to be installed by the user.

# Android Manifest

- Another important function that the manifest provides is the setting of minimum API levels
  - The minimum API level states that an android device
  - This is enforced for compatibility reasons. Any device running a lower API level will refuse to run the application
    - While there are support libraries to support older devices it introduces complication.

# Android Manifest

- The final purpose of the Manifest file is to list all the activities, services, and components that make up the application
  - All components must be visible to the Android OS
  - If an application attempts to start an activity or service that is not listed in the manifest it will be shut down immediately
  - For security reasons which will be explored later

# The layout/ Directory

- Here is where the layouts for all activities and fragments are stored in android
  - All layout files are written in XML meaning they will compress well.
  - Also generating layouts using Java code directly is possible
    - However it is tedious error prone and does not provide as many options as the XML version does.

# The layout/ Directory

- Layouts do not have to be restricted to a single activity or fragment
  - They may be shared
- It is also possible to split out large layouts into smaller files to reduce complexity
- Another reason you should use the XML layouts here is that all XML layouts will use the layout inflator service
  - Very quick at building layouts from XML

# The menu/ directory

- All menus are stored in this directory (not present by default)
- Like layouts all menus are written in XML text format to take advantage of text compression
  - Also like layouts menus can be shared amongst multiple activities
  - Contain a full menu ordering system and the ability to add items to the action bar

# The values/ directories

- Here is where small non code resources collections are stored.
- All of the files here are also XML based for maximum compression
- The collections that are maintained here are things like strings and colours
  - There are three files here by default

# The values/ directories

- Strings.xml contains a list of strings that are used throughout the application
  - It is recommended that you store strings here and reference them in your application as a mistake in a string can be easily fixed by modifying the xml file instead of tracking all instances within your code resources
- The other reason this is useful
  - Is supporting multiplie languages. For a different language specify a different strings.xml with all the same ids but different strings

# Activities

- All applications in Android are required to fullscreen on the device.
  - To make the most of the space that is available.
- Thus it is not possible to use a single activity to represent the full functionality of your application.
  - Thus you require multiple activitities and methods from transfering from one activitiy to the next.

# Intents

- To link from one activity to the next we use the intent system.
- An intent asks the android system to start a new activity for this application
- An intent may have data attached
- An intent may also expect a result to be returned.

# Intents

- There are two different classes of intent however
- An explicit intent: where you know the exact name of the activity that you wish to start
  - Generally used within your own application
- Or an implicit intent

# Intents

- An implicit intent states to the android system that we have some data to display
  - However, we cannot render it ourselves, but we would like android to find an application to render it for us
- Depends entirely on a compatible application being present
  - May or may not resolve

# Intents

- This is the real power of the Android system
- Applications pick a specific job and do it well.
  - If then need support for youtube or maps they can hand that data to a seperate application
  - Instead of implementing that behaviour itself.

# Intents

- Thus when you are looking for functionalility
  - You can try and use a seperate application instead to do the functionality
  - Reduces the amount of code for you to write
- Increases reliability

# General advice on application structure

- Firstly try and use as many of the non code resources possible
  - As stated before this will result in the smallest size of your application.
- Secondly if you are supporting a wide range of screen sizes use multiple layout files
  - Can restructure the application without affecting code too much

# General advice on application structure

- If you find that your mipmaps are too large and taking up much space
  - Try putting custom views in their place
  - May be more code but a single mathematical description will scale to all sizes
  - And will remove the relevant drawables from your package.

# General advice on application structure

- Take full advantage of strings.xml
  - May seem like more effort to separate strings and setup references
  - However, if an error is spotted it is easy to correct
  - And multiple languages are easy to support this way.

# Java/ directory

- The java directory will seem a little odd at first as it looks to contain two copies of all of your code.
  - However one of these will be labelled with the words (androidTest) written after it.
  - This is a set of JUnit tests that you can use to unit test your code instead of running the application over and over and verifying by hand

# Java/ directory

- If you haven’t learned unit testing yet I suggest you do so
  - Used nearly everywhere.
- Of course the other java directory with no labelling is where your actual android code resides.