FIT2081 – Mobile Development

# Week 2 – Basic Components

* Minimum SDK version determines the lowest level of Android that your app will run on

## API fragmentation

* Fragment: a modular section of an activity 🡪 can be reused
* Relate to the updates of software
* Forward compatibility
  + Old apps running on new platform versions
  + Android apps are generally forward-compatible with new versions of the Android platform 🡪 most framework API are additive
* Backward compatibility:
  + New apps running on old platform
  + Can be solved by replacing existing API parts

## Android Components

Read more: <https://developer.android.com/guide/components/fundamentals.html>

### Activities

* Entry point for interacting with the user
* Represents a single screen with a user interface
* An activity facilitates the following key interactions between system and app:
  + Keeping track of what user currently cares about (what is on the screen)
  + Knowing the previously used processes contain things the user may return to (stopped activities)
  + Helping app handle having its process killed so the user can return to activities with their previous state restored
  + Providing a way for apps to implement user flows between each other & for the system to coordinate these flows
* The same activity can be started from different apps
* Activated by intents

### Services

* A general-purpose entry point for keeping an app **running** in the **background** for all kinds of reasons (do not have an user interface)
  + Playing music in the background
* 2 lifecycle:
  + Run until it finishes
  + Run as long as its bound-to process is still running
* 2 types of started service:
  + User is aware
  + User is not aware
* Activated by intents

### Content Providers

* Implement a mechanism to share data between applications
* Can store data in the file system
* Access to data is provided via a Universal Resource Identifier (URI)
* Data can be shared in the form of a file or an entire SQLite database
* **Content Resolver:**
  + The single, global instance in your application that provide access to your (other applications’) content providers
  + Includes: CRUD (create, delete, update and delete) methods
* Activate by request from content resolver

### Broadcast Receivers

* Component that enables the system to deliver events to the app outside of regular user flow
* Can deliver to apps that are not running
* Usually used as a system notification
* Activated by intents

### Related

#### Intent

* Used for activating 3 out of 4 components: service, activities and broadcast receivers

#### Manifest file

* XML file
* Includes:
  + A declaration of all components in the application
  + If a component is not declared the system can’t see it
* Primary task: informing the system about the app’s component
* Other tasks:
  + Identify any user permission the app requires
  + Declare minimum API level required by the app
  + Declare hardware and software features used or required by the app (camera, Bluetooth service)
  + Declare API libraries the app needs to be linked against (other than the Android framework API)

#### Resources

* Include strings, images, fonts, colours that appear in the user interface together with the XML representation of the user interface layouts
* By default, these files are stored in the /res
* Using resources make it easier to update various characteristics of the app without modifying the code and by providing sets of alternative resources – enables you to optimise the app

#### Context

* Interface to global information about an application environment
* When an application is compiled, a class named R is created that contains references to the application resources

# Week 3 – Life Cycle

# Week 4 – View & Layout

* Android UIs are made of a hierarchy of View objects

## View:

* + Basic building block for UI components
  + Occupies a rectangular area and responsible for drawing and event handling
  + Base class widget, used in creating interactive UI components (button, text fields, etc…)

## View group:

* + Base class for layout and view container, contains many views (children)
  + Define **ViewGroup.LayoutParams** class which serves as the base class for layout parameters
  + Allow views to be nested 🡪 represent as a hierarchy (HTML and XML alike)
  + The tree is call **Layout** (responsible for managing the size, position and behaviour of all the Views it contains)

## ViewGroup.LayoutParams:

* + Used by views to tell their parents how they want to be layout
  + Check [ViewGroup Layout Attribute](https://developer.android.com/reference/android/R.styleable.html#ViewGroup_Layout) for all the supported child attributes
  + Base LayoutParams class just describe how big the view wants to be for width and height
    - FILL\_PARENT (MATCH\_PARRENT in API 8 and above) 🡪 wants to be as big as parent minus paddings
    - WRAP\_CONTENT 🡪 wants to be big enough to enclose its content (plus padding)
    - An exact number
  + There are subclasses for this class that adds extra attributes

## Layout Parameters

* + XML layout attributes named layout\_something define layout parameters for the views that its ViewGroup contains

## Creating UIs

* 2 ways to declare a layout:
  + XML
  + **Instantiate** layout elements at runtime (Java)
* Java or XML:
  + XML:
    - enables you to better separate the presentation of your application from the code that control its behaviours
    - External so you can modify or adapt it without having to modify your source code and recompile
    - Easier to visualise the structure of your UI
  + Java:
    - if you really want this 🡪 refer to ViewGroup and View class references
    - check out JavaLayout app
* Personal opinion: Java is more complicated and confusing when dealing with layout 🡪 have to use parameters and stuff

## Layout types

### ViewGroup

* Has several Layout direct subclasses
  + CoordinatorLayout, FrameLayout, GridLayout, LinearLayout, RelativeLayout
* Has several Layout indirect classes
  + TableLayout
* ViewGroup is also a View 🡪 can be contained in another ViewGroup
  + Should aim for optimisation
  + Google introduced ConstraintLayout for optimisation sake

### View containers

* Many direct and indirect View container subclasses that can be contained in another view ( be part of a UI’s View hierarchy). E.g. Toolbar

### ConstraintLayout

* A new direct subclass found in the support library
  + Use SDK Manager to download the Android Support Repo 🡪 “ConstraintLayout for Android” and “Solver for ConstraintLayout”
  + Edit Gradle Scripts to include the necessary dependencies
* Compatible with API level 9
* Basically like a RelativeLayout but:
  + Views are attached to the layout sides or horizontal and vertical guidelines (virtual layout sides) and other Views by software analogues of springs
  + These springs can expand and collapse depending on the viewport of the device (including current orientation)
  + The tension can be determined by a percentage
  + Hard margins can be specified

## Styles

* A collection of attributes that specify the look and format for a view or window
* A style is defined in an XML resource that is separate from the XML that specifies the layout
* Style Inheritance:
  + Parent attribute in the <style> elements lets you specify a style from which your style should inherit attributes 🡪 use this to inherit attributes from an existing style and define only the attributes that you want to change or add
  + Similar to CSS

## Themes

* Style applied to an entire Activity or app [in the app’s Manifest file]
* Apply the Style to a set of attributes

## Material Design

* Comprehensive guide for visual, motion and interaction design across platforms and devices
* To use material design 🡪 follow guidelines defined in the material design specification and use the new components and functionality available in Android 5.0 (API level > 21 )
* [Tutorial](https://developer.android.com/guide/topics/ui/look-and-feel)
* Compatibility complications: cannot use Material theme with standard Activity super class (not able for maximum backward compatibility)
  + Works for Activities that are subclass of Activity
* Use v7 for backward compatibility to API 9 (2.3)
* Deeper MD:
  + [Overview](https://developer.android.com/design/material/index.html)
  + [Getting Started](https://developer.android.com/training/material/get-started.html)
  + [Training overview](https://developer.android.com/training/material/index.html)
  + [Customising MD theme](https://developer.android.com/training/material/theme.html)
  + [Compatibility](https://developer.android.com/training/material/compatibility.html)

## Action Bar vs AppBar

* App bar was formerly known as action bar, used for branding, navigation, search and actions
* No UI components, it is a design concept 🡪 more [info](https://developer.android.com/design/patterns/actionbar.html)
* Pre-lollipop: part of default theme (not a widget)
* Post-lollipop: no longer part of default theme 🡪 work more like a toolbar widget
* No UI component but there is a UI component called Toolbar and a confusing class named ActionBar 🡪 which addresses whatever is the AppBar/Action Bar for an Activity’s UI (could be implemented as an AppBar or Toolbar)
* android.support.v7.widget.Toolbar class
  + “A Toolbar is a generalization of action bars for use within application layouts. While an action bar is traditionally part of an Activity's opaque window decor controlled by the framework, a Toolbar may be placed at any arbitrary level of nesting within a view hierarchy. An application may choose to designate a Toolbar as the action bar for an Activity using the setSupportActionBar() method.”
* setSupportActionBar(Toolbar toolbar) Method
  + When set to a non-null value the getActionBar() method will return an ActionBar object that can be used to control the given toolbar as if it were a traditional window decor action bar
  + In addition to the Toolbar’s methods

### Using an ActionBar as your App Bar (not preferred)

* Use a theme which does not contain “.NoActionBar”
* The Action bar will appear without any coding but use getSupportActionBar() to gain a reference to it so it can manipulated
* There are API level differences as features were incrementally added

### Using a Toolbar as your App Bar (preferred)

* Use a theme which DOES contain “.NoActionBar” so there is no ActionBar
  + Style item elements in a style’s XML definition
* No API differences
* Use an android.support.v7.widget.Toolbar instead and use setSupportActionBar (Toolbar toolbar) to set the Toolbar to act as the Action Bar for this Activity window.

### Relevant videos:

* + [Toolbar: How do they work?](https://www.youtube.com/watch?annotation_id=annotation_2860510657&feature=iv&index=5&list=PLWz5rJ2EKKc-lJo_RGGXL2Psr8vVCTWjM&src_vid=THadGrPeSJM&v=kmUGLURRPkI)
  + [AppBarLayout and scrolling gestures](https://www.youtube.com/watch?v=THadGrPeSJM&ab_channel=AndroidDevelopers)