# MET CS 683
# Mobile Application Development with Android

Security
Graphics and Animation

# Android Platform Security

- Security Objectives:
  - Protect app and user data
  - Protect system resources (including the network)
  - Provide app isolation from the system, other apps, and from the user
- Key security features:
  - Robust security at the OS level through the Linux kernel
  - Mandatory app sandbox for all apps
  - Secure interprocess communication
  - App signing
  - App-defined and user-granted permissions
- https://source.android.com/security

# Store Data

- Internal storage
  - Use only MODE_PRIVATE (in older APIs)
  - Each application has its own data folder, which is isolated from other application through different UID&GID. It can only be accessed by its own application.
  - Shared preferences files are text files.
  - The internal storage is encrypted (Android 6 or above)
  - For highly sensitive data, additional protection is needed such as encryption or crypto provided by the security library.

```
drwx------  4 u0_a96 u0_a96 4096 2018-04-28 17:12 edu.bu.myapplication1
drwx------  6 u0_a92 u0_a92 4096 2018-04-25 13:28 edu.bu.projectportal
drwx------  4 u0_a89 u0_a89 4096 2018-04-22 04:50 edu.bu.testmfcc
drwx------  4 u0_a95 u0_a95 4096 2018-04-28 17:05 edu.bu.threadhandlerexample
drwx------  4 u0_a94 u0_a94 4096 2018-04-26 04:50 edu.bu.useprojectportalprovider
drwx------  4 u0_a86 u0_a86 4096 2018-04-03 19:39 edu.bu.widgetsexplore
```

# Store Data

- External storage
  - Universal accessible by all applications.
  - Not trusted and may not be encrypted at all
  - Never store any sensitive data directly
  - Not good for dynamic loading
  - Use security library to provide some protection
- Content providers
  - Mostly use it to export data to other applications
  - `Android:exported` is false by default in Android API 17

# Android Permissions

- The principle of least privilege
- Protect system resources
  - Cost-sensitive services, such as telephony, SMS/MMS, Network/Data, In-App Billing or NFC Access.
  - Personal information, such as the phone book or the calendar.
  - Sensitive data Input devices, such as GPS,  camera, or microphone.
  - Device metadata, such as system logs, browser history, phone numbers, or hardware / network identification information.
- Protect communication between applications.
  - When executing certain security sensitive functions
  - When starting an activity
  - When starting or binding a service
  - When sending or receiving broadcasts
  - When accessing a content provider

# Use Permissions

- Request permissions
  - Each application need to declare all permissions explicitly with a <uses-permission> element in the manifest file. Each permission is identified by a unique label, such as INTERNET and ACCESS_FINE_LOCATION.

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

  - Need to dynamically request dangerous permissions explicitly in the code when using it for the first time using requestPermissions() method.(Android 6 or above)The user can deny a permission and still continue to use the app.

# Installation Time Permission vs Runtime Permission

- All permissions in use need to be declared in the manifest file. Normal permissions are automatically granted without the need of the user's approval.
- Before Android 6, installation time permission mechanism is used
  - All permissions including dangerous permissions are granted if the user agrees to install the application at the installation time. Otherwise, the installation is canceled.
- In Android 6 and above,
  - The user can turn on and off individual permission for all installed apps.
  - For apps which is built for the older version, when used with Android 6. permissions are granted in the installation time for backwards compatibility.
  - For apps which is built for Android 6 and above, dangerous permissions declared in the manifest are not granted in the installation time, instead, the app need to request them explicitly in the runtime using requestPermissions() method. The user can deny a permission and still continue to use the app.

# Request Permission Dynamically

- `ActivityCompat.requestPermissions()`
- `ContextCompat.checkSelfPermission()`
- `ActivityCompat.shouldShowRequestPermissionRationale()`
- `onRequestPermissionsResult()`

# Define Custom Permissions

- Application can also define customized permissions with the <permission> element in the manifest file.

```
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.myapp" >

    <permission
      android:name="com.example.myapp.permission.DEADLY_ACTIVITY"
      android:label="@string/permlab_deadlyActivity"
      android:description="@string/permdesc_deadlyActivity"
      android:permissionGroup="android.permission-group.COST_MONEY"
      android:protectionLevel="dangerous" />
    ...
</manifest>
```

# Impose Permission

- A component can use “android:permission” attributes to restrict which applications can use this component(send intents to it)

```
<service
   android:name=".services.MyService"
   android:enabled="true"
   android:exported="true"
   android:permission="android.permission.ACCESS_FINE_LOCATIO,
   com.example.myapp.permission.DEADLY_ACTIVITY"
   />
```

- Can also programmatically check permission using checkCallingPermission()

# Secure Inter-Component Communication

- The core components of an application, such as its activities, services, and broadcast receivers, are activated by intents
- An Intent is a message object used for inter component communication.
- Intent Filter:
  - Defines what kind of intents can receive by action strings.
  - A component can have any number of filters, each one describing a different capability defined by action strings.

# Secure Inter-Component Communication

- Explicit Intents: specifying the receiving component.
- Implicit Intents: declare a general action without specifying the a specific receiving component.
  - When using an implicit intent, the Android system locates an appropriate component that can respond to the intent, launches a new instance of the component if one is needed, and passes it the Intent object
  - Explicitly show an app chooser if there are multiple apps that can match
- Pending Intents
  - A wrapper around an Intent object.
  - A token given to a foreign application (e.g. NotificationManager, AlarmManager, Home Screen AppWidgetManager, or other 3rd party applications), which allows the foreign application to use the sending application permissions to send the wrapped intent.

# Open Component

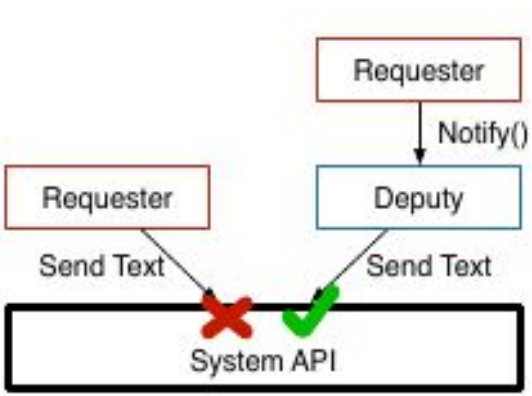
- An open component is a component that be reached by other applications through intents.
- In general, a component is private by default unless the exported attribute is true or an intent filter is defined.
- An open component can be exploited by malicious applications if not properly protected.

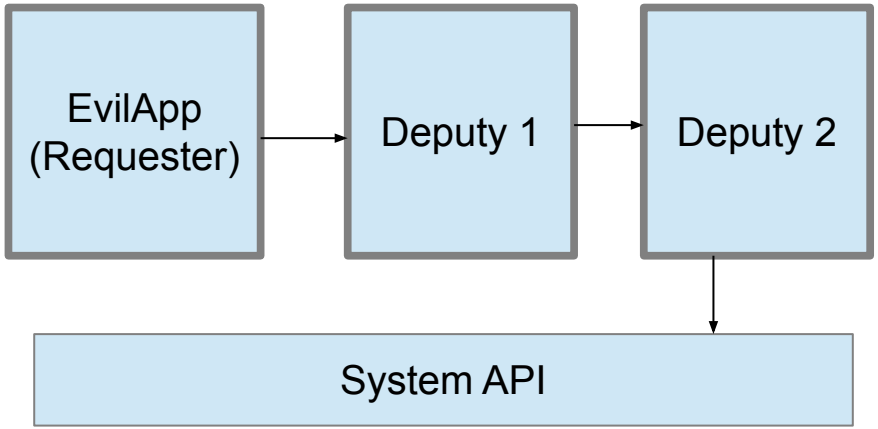
```
<activity
   android:name=".services.MyActivity"
   android:exported="true"
     android:permission=””/>
<service
   android:name=".services.MyService"
   android:enabled="true"
   />
<receiver
   android:name=".services.MyReceiver"
   />
```

```
<activity
   android:name=".activities.MainActivity"
   android:label="@string/app_name"
   android:theme="@style/AppTheme.NoActionBar">
   <intent-filter>
       <action
android:name="android.intent.action.MAIN" />
       <category
android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

# Privilege Escalation

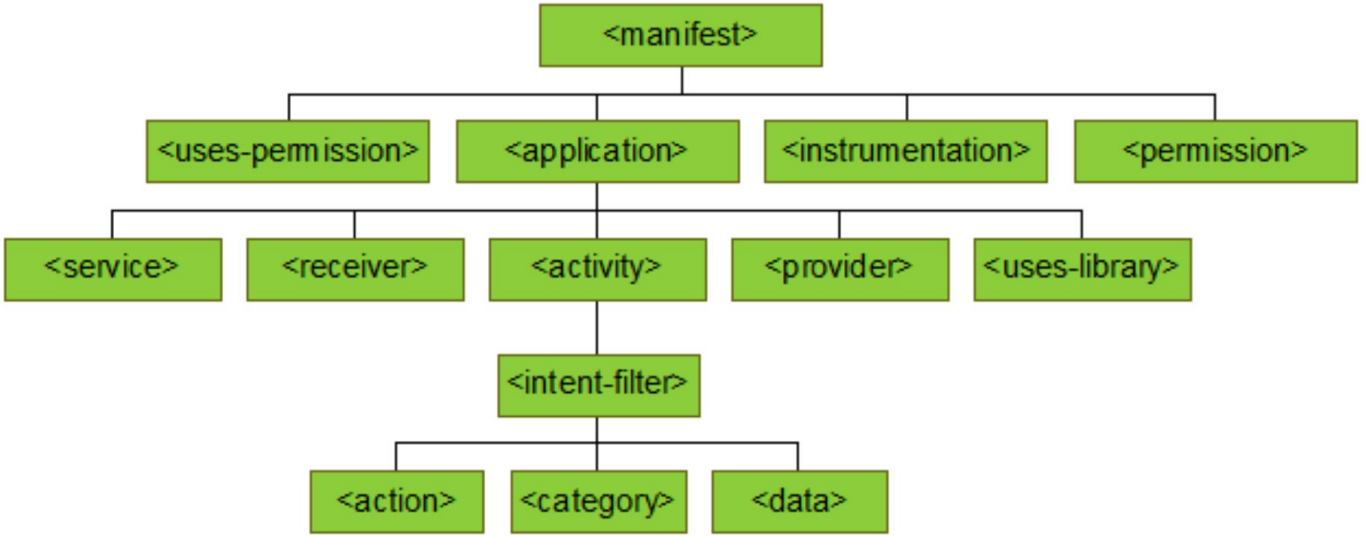
- Permission re-delegation occurs when an application with a permission performs a privileged task on behalf of an application without that permission.



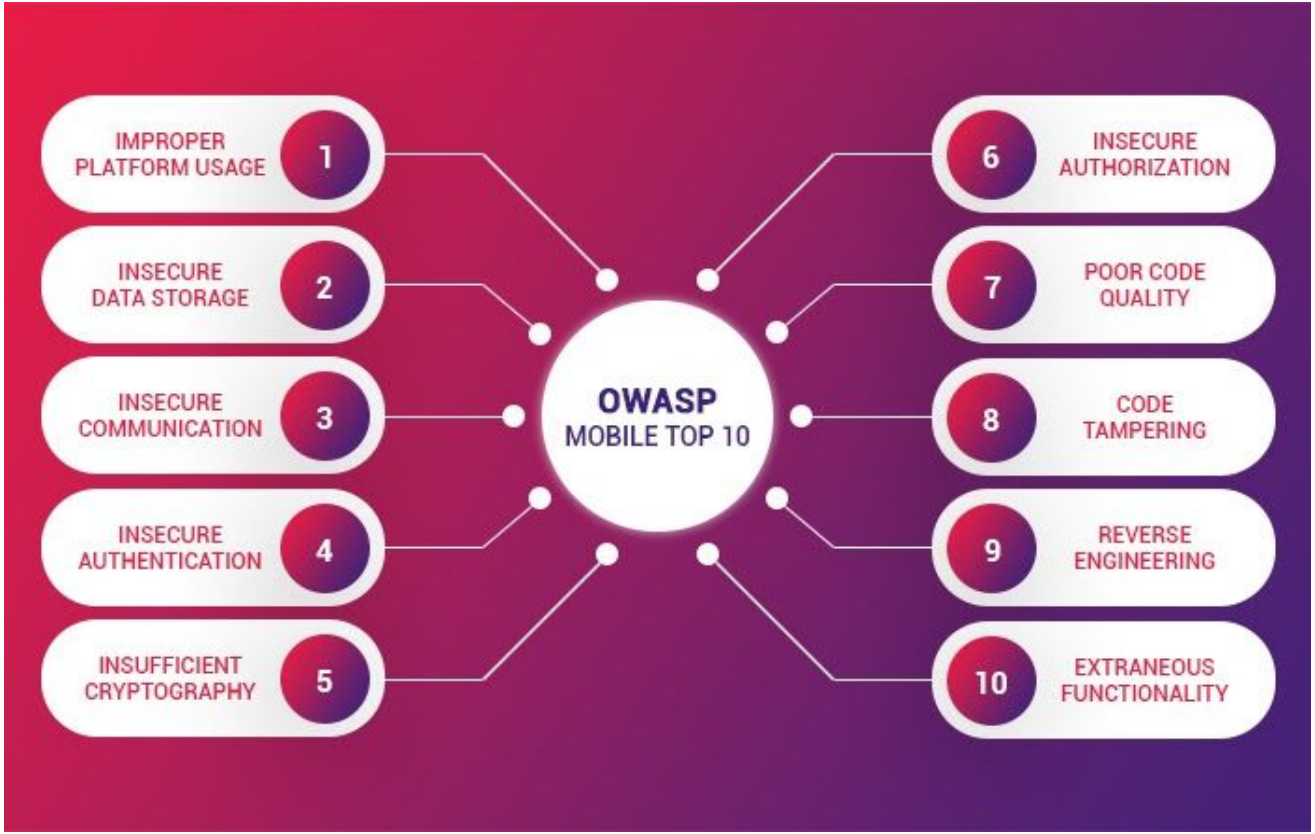



# A Brief Review of Manifest File

- Every application must have an AndroidManifest.xml file (with precisely that name) in its root directory
- The manifest file provides essential information about your app to the Android system, which the system must have before it can run any of the app's code



# OWASP Mobile Top 10




https://www.appsealing.com/owasp-mobile-top-10-a-comprehensive-guide-for-mobile-developers-to-counter-risks/

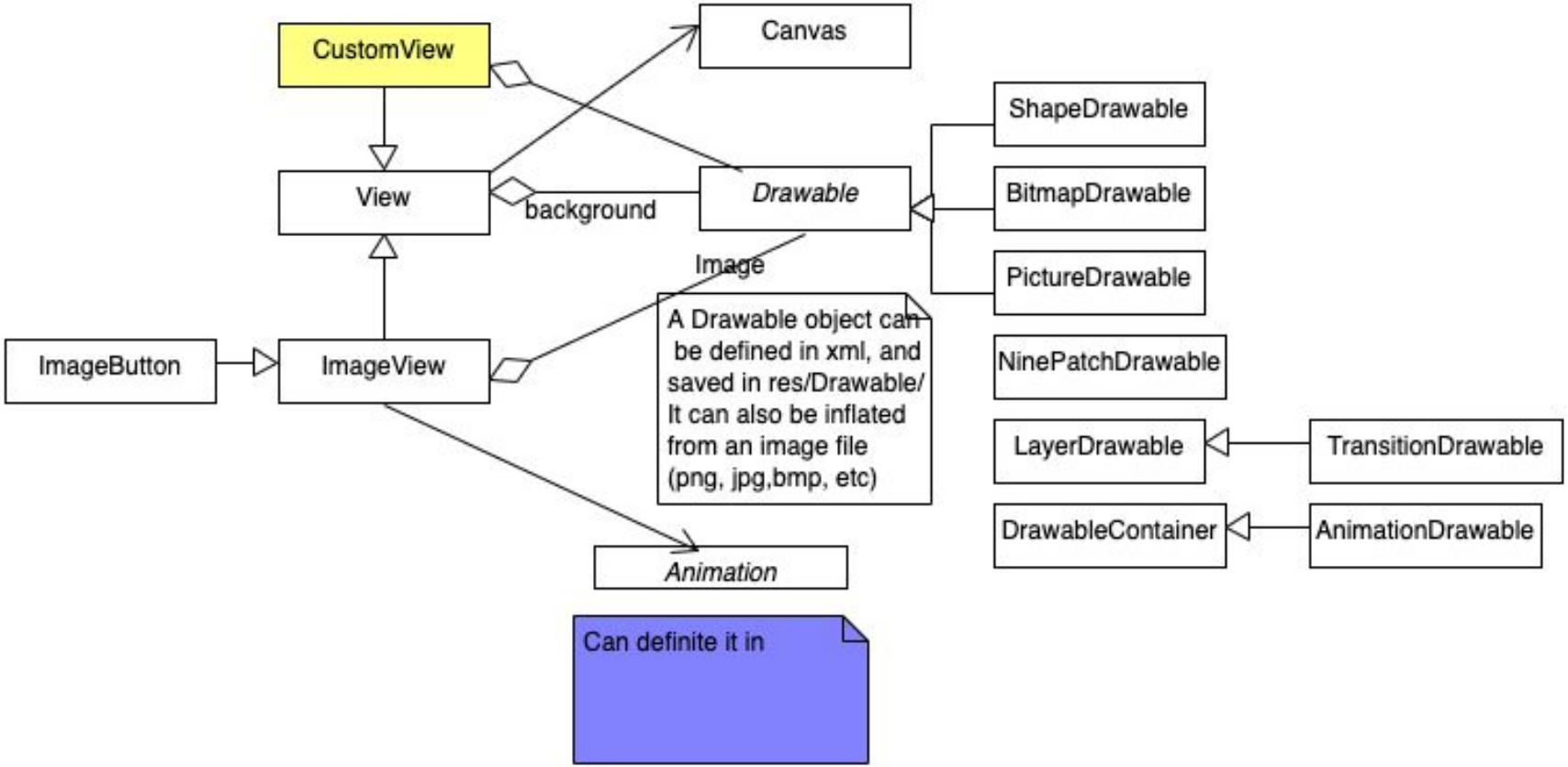
# 2D Graphics

- Simple graphics: display static graphic or predefined animation, within a relatively static app that don’t need to change dynamically.
  - Draw into a View object from layout
    - handled by the system's normal View hierarchy
    - define the graphics to go inside the View
  - Common packages used for 2D drawing and animating: android.graphics.drawable, android.view.animation
- Interactive game (with the need of constantly redrawing) and 3D rendering.
  - Draw directly to a Canvas.
    - Call appropriate class's draw() (passing Canvas)
    - or a Canvas draw...() method (e.g.,
      - In the same thread as UI Activity, create custom View component in layout, call invalidate(), then handle onDraw() (invalidate() forces a View)
      - In separate thread, managing a SurfaceView ; perform draws to the Canvas as fast as thread can (no need for invalidate())

# Drawable

- Abstraction for "something that can be drawn"
- Only for display, cannot receive events or interact with the user.
  - The simplest case is a graphical file (bitmap), which would be represented in Android via a BitmapDrawable class.
- Drawables can be stored as individual files in one of the res/drawable folders.
  - Bitmap files (png, jpg): usually stored for different resolutions in the -mdpi, -hdpi, -xhdpi, -xxhdpi subfolders of res/drawable. The Android system selects the correct one automatically based on the device configuration.
  - XML drawables: used to describe shapes (color, border, gradient), state, transitions and more.
  - 9-patch graphics are used to define which part of a graphic should be stretched if the view which uses this graphic is larger than the graphic.
- Drawables can also be created directly in the Java code. Every object which implements Drawable can be used as a Drawable in code.

# Class Diagram



# Drawable

- An **abstract** class: android.graphics.drawable.Drawable
- SubClasses:
  - BitmapDrawable
  - ShapeDrawable
  - PictureDrawable
  - NinePatchDrawable
  - LayerDrawable
    - TransitionDrawable
  - DrawableContainer
    - AnimiationDrawable
- Can also extend the class to custom Drawable objects
- Drawable objects are then used by view objects.

# Drawable

- In res/drawable folder
  - Image files: png, jpg, etc
  - Self defined XML file
- In res/layout
  - Refer to the drawable object in the XML layout using its resource id (filename without the extension name)
  - Layout can be inflated into the view
- In Java code
  - Load the drawable object in res/drawable using its resource id into a view object such as an ImageView object
  - Create a drawable object using the resource id of the drawable object in res/drawable
  - Load the drawable object created in the Java code into view or customized view objects.

# Using Drawable in Views

- Drawables stored in the res/drawable are referred via its resource id, which is the filename without the file extension.
  - In XML Layout, it is referred via @drawable/resourceid
  - In Java code, it is referred via R.drawable.resourceid resource ID as input parameter.
- Most view can use a drawable to set its background. Imageviews also use a drawable as its image source.
  - In XML Layout, the resource id can be used in some attributes such as android:background or android:src.
  - In Java code, the resource id can be passed to views, e.g. through **setBackgroundResource(**int resId) or **setImageResource**(int resId)
  - Or create a drawable object and pass to views, e.g. through setBackground(Drawable drawable)

# Example: LayerDrawable & TransitionDrawable

- LayerDrawable is a Drawable that manages an array of other Drawables.
- These are drawn in the array order, the element with largest index being drawn on top.
- TransitionDrawable is a subclass of LayerDrawable
- It provides a cross-fade between the first and second layer, is definable in XML and executed with startTransition().
- https://developer.android.com/reference/android/graphics/drawable/TransitionDrawable.html

# Example: ShapeDrawable

- To dynamically draw 2D graphics, a ShapeDrawable object can be used programmatically draw primitive shapes and style them
- Extends the Drawable class, so it can be used wherever the Drawable is expected (e.g., background of a View, set with setBackgroundDrawable()).
- Can also draw shape as its own custom View, to be added to layout.
- Can create a subclass of View that draws the ShapeDrawable during View.onDraw() execution. (ShapeDrawable has its own draw() method)
- Can draw the shape programmatically in an Activity.

# Example

- To draw this from the XML layout instead of from the Activity, add a CustomDrawable element to the XML:

```
<com.example.shapedrawable.CustomDrawableView
android:layout_width="fill_parent"
android:layout_height="wrap_content" />
```

- CustomDrawable must override View(Context, AttributeSet) constructor
  - called when instantiating a View via “inflation” from XML.

# Using Canvas

- When writing an app requiring specialized drawing and/or control the animation of graphics
- A canvas serves as a pretense, or interface, to the actual surface upon which your graphics are drawn—
- You can perform your draw operations to the canvas.
- Drawing actually performed on an underlying Bitmap, which is placed into the window
- Android framework provides canvas for View objects. You can create a subclass of View and implement the onDraw(Canvas) callback
- The Android fram**ework only calls onDraw(Canvas) when necessary. To redraw your app, you must first call invalidate().** Then Android calls your view's onDraw(Canvas) method, though the call isn't guaranteed to be instantaneous.

# Using Canvas

- Can also acquire a Canvas from SurfaceHolder.lockCanvas(), when dealing with a SurfaceView object. The goal is to offer this drawing surface to an app's worker thread.
- Canvas has drawing methods: drawBitmap(...), drawRect(...), drawText(...), …
- Other drawing classes also have draw(canvas) methods
  - E.g., Drawable objects for placement on the Canvas
  - Drawable has its own draw() method with Canvas argument
- Customize View's onDraw(canvas) method using various drawing methods of canvas or Drawables' draw() method, passing canvas to drawables.
- If need to create new Canvas, define its Bitmap first

```
Bitmap b = Bitmap.createBitmap(100, 100, Bitmap.Config.ARGB_8888);
Canvas c = new Canvas(b);
```

# Property Animation

- Property Animation: changes a property's (a field in an object) value over a specified length of time.
  - Java classes:  ObjectAnimator, ValueAnimator, AnimatorSet
  - Resource file location: res/animator/filename.xml
  - Reference: @[package:]animator/filename, R.animator.filename

# View Animation

- Tween animation: performing a series of transformations on a single image with an Animation (such as fading, moving, stretching)
  - Java classes:  Animation
  - Resource file location: res/anim/filename.xml
  - Reference: @[package:]anim/filename(in XML),  R.anim.filename(in Java)
- Frame animation: showing a sequence of images in order (like a film) with an AnimationDrawable.
  - Java classes:  AnimationDrawable
  - Resource file location: res/drawable/filename.xml
  - Reference: @[package:]drawable/filename(in XML), R.drawable.filename(in Java)
- https://developer.android.com/guide/topics/resources/animation-resource.html#Property

# Question?