Learning Unit 3

Android fundamentals

MOAATH ALRAJAB

Objectives:

- Take a deeper look at Hello World App
  - Change app name and add more controls
  - Understand resources and the manifest file
  - Enable an event and add a Toast
- Discuss Android fundamental building blocks and components
- Understand the Activity lifecycle
- Discuss MVC and Android Resources

Fundamentals of Android Application

The very basic four fundamental building blocks from which all Android applications are built are:

- **Activities**
- **Services**
- **Broadcast Receivers**
- **Content providers**

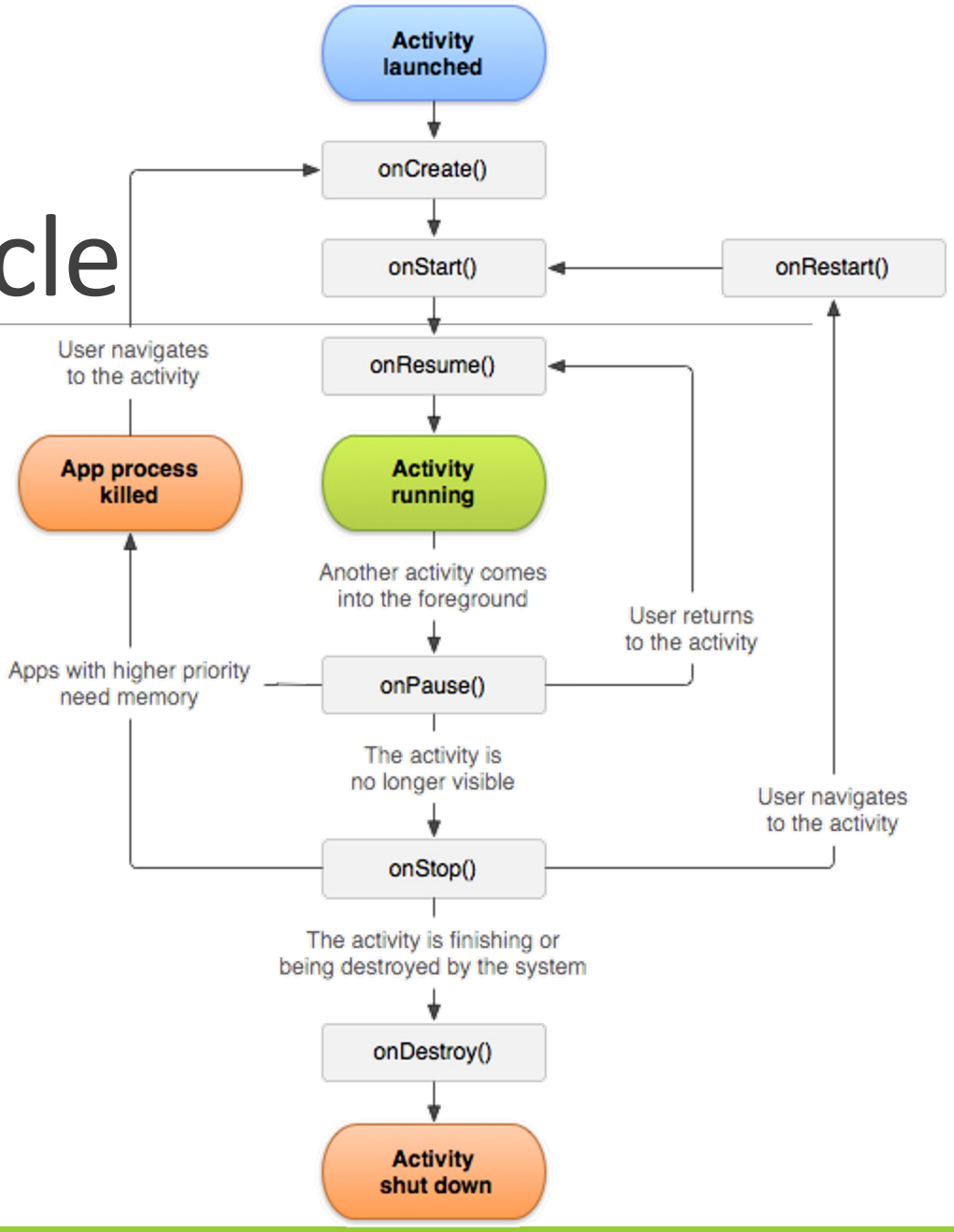
Others such as Intent, ***Resource*** Externalization, and Notifications

The focus in this learning unit is Activity and Resources.

Activities

- Activity class is designed to present a GUI.
- This class supports a single focused screen that the user can use such as dialing a phone number or entering contact information and so on.
- Activities are public classes, inheriting from the android.app.Activity base class
- This class will be discussed further. But first let’s check the lifecycle of an activity.

Activity Lifecycle

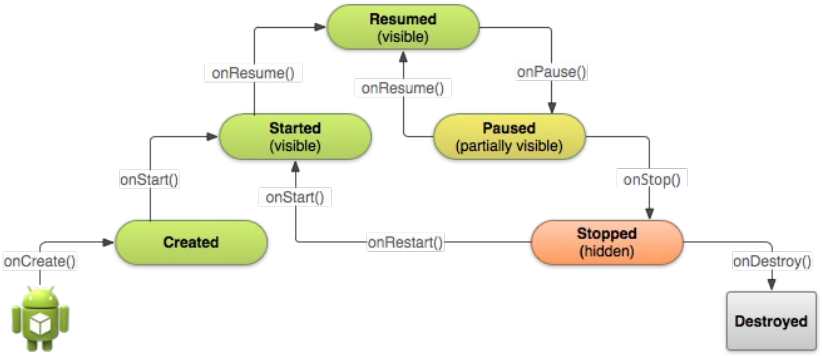


Activity Lifecycle – cont.

- onCreate – called when first created
(and when portrait/landscape changes)

- onStart – called just before activity
becomes visible on screen

- onResume – whenever activity comes
to the foreground



- onPause – when activity is stopped and
no longer visible, used to suspend resource intensive actions

- onStop – when activity is no longer visible or is being destroyed

- onDestroy – when activity is completed and is going to be destroyed

Service

Runs in the background

Perform long running operations

Support interaction with remote processes.

Example: backup application, running a music track and so on.

Broadcast Receivers

- Broadcast receiver's listen for and respond to events.
- This class plays the role of the subscriber in the publish/subscribe pattern.
- In Android, events are represented by the intent class.
- Publishers create these intents and then broadcast them using a method like sendBroadcast(Intent, String).
- Now, once broadcasted, these intents are then routed to the broadcast
- receivers that have subscribed to, or registered for, those specific intents. At which point, they can respond to the event.

Content Providers

- Store and share data across applications
- Handle requests
- Keep tracks of data

Example: the browser application uses the contentprovider class.

Building Android Application

The Android build process provides project and module build settings so that your Android modules are compiled and packaged into .apk files. The apk file for each app contains all of the information necessary to run your application on a device or emulator, such as compiled .dex files (.class files converted to Dalvik byte code), a binary version of the AndroidManifest.xml file, compiled resources (resources.arsc) and un-compiled resource files for your application.

http://developer.android.com/tools/building/index.html

Android PacKage (APK)

- Each project should be packaged as .apk to run.
- Use Gradle to compile, build and create apps
- Every application must have an AndroidManifest.xml file (with precisely that name) in its root director which:
  - names the Java package for the application.
  - describes the components of the application
  - also declares all permissions
  - declares the minimum level of the Android API that the application requires.
  - lists the libraries that the application must be linked against.

http://developer.android.com/guide/topics/manifest/manifest-intro.html

Android PacKage (APK) cont.

- Each Android App is secured by the following features:
  - Android OS which is a multi-user Linux system and each app is a unique user.
  - Each app is assigned a unique Linux user ID with its assigned permission (files…)
  - Each app has a its own copy of the virtual machine that runs the dex files.
  - Every app has its own thread. The android system starts the process when an app is executed.
  - Android is an OS that implement the ***principle of least privilege*** (each app access its components and what else permitted).

http://developer.android.com/guide/topics/manifest/manifest-intro.html

Creating Android App process

Defining resources

Implementing classes

Packaging application

Deploying the app.

A well planned app should define resources such as images, media, layouts and so on. This allows modifying application without recoding.

One important resource is the String class.

Device Compatibility

Android is designed to run on many different types of devices, from phones to tablets and televisions. The range of devices provides a huge potential audience for your app. In order for your app to be successful on all these devices, it should tolerate some feature variability and provide a flexible user interface that adapts to different screen configurations.

There are two types of compatibility:

1. **device compatibility** and
2. **app compatibility**.

Device Compatible

Android is an open source project, any hardware manufacturer can build a device that runs the Android operating system. Yet, a device is "Android compatible" only if it can correctly run apps written for the Android execution environment.

Google Play Store has only the compatible devices.

App Compatible

Android supports a variety of features your app can leverage through platform APIs. Some features are hardware-based (such as a compass sensor), some are software-based (such as app widgets), and some are dependent on the platform version. Such as **screen resolution**, **features** and **platform** compatibility.

// For feature compatibility

```
<manifest ... >
    <uses-feature android:name="android.hardware.sensor.compass"
                  android:required="true" />
    ...
</manifest>
```

// For platform compatibility

```
<manifest ... >
    <uses-sdk android:minSdkVersion="14" android:targetSdkVersion="19" />
    ...
</manifest>
```

Manifest File

Name space

Package

versionCode/versionName

<uses-sdk>
- minSdkVersion
- targetSdkVersion
- maxSdkVersion (optional)

<application> - identifies important items for the application

<activity> - identifies activities and their properties

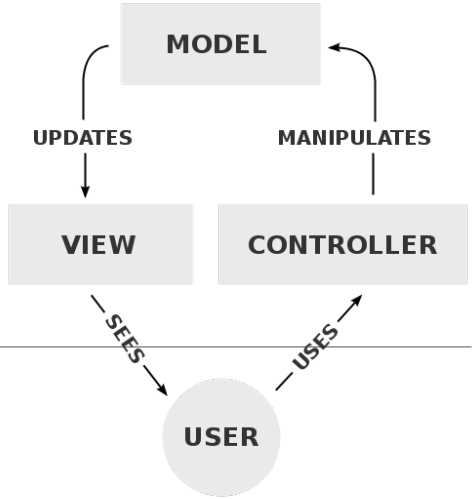
Other commonly used tags: <service>, <receiver>, <provider>, <uses-permission>

Manifest File – You need to visit the link below for more info

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.defaultexample"
   android:versionCode="1"
   android:versionName="1.0" >
   <uses-sdk
      android:minSdkVersion="8"
      android:targetSdkVersion="17" />
   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
         android:name="com.example.defaultexample.MainActivity"
         android:label="@string/app_name" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```

http://developer.android.com/guide/topics/manifest/manifest-intro.html

MVC and Android



What is MVC:

It is apparent that the user interface logic
tends to change more often than the business logic,
the mobile, desktop, and Web developers needed a much easier way of separating UI from actual code (the brain).
This separation is called MVC.

Model — the data layer, responsible for managing the business logic and handling network or database API.

- In simple words, what to show

View — the UI layer — a visualization of the data from the Model.

- In simple words, How to show

Controller — the logic layer, gets notified of the user's behavior and updates the Model as needed.

- In simple words, compute, events, validate and so on

MVC and Android cont.

MVC is a concept rather than a programming language or framework. The way you implement varies.

How Should MVC Be Applied in Android

The Activities, Fragments and Views should be the **Views** in the MVC world. (derived from android.view.View so they are different in Android from iOS and so on)

The **Controllers** should be separate classes that don't extend or use any Android class (controllers should be platform/app independents),

and same for the **Models**. Models should be platform independents.

Resources – definition

Resources are the data that your application uses:

- *Maintain them independently*:

  Resources such as images and strings should be separated from your application code, so that you can maintain them independently.

- *Widen device support*:

  Externalizing your resources also allows you to provide alternative resources that support specific device configurations such as different languages or screen sizes, which becomes increasingly important as more Android-powered devices become available with different configurations

Resources – by nature

Default resources

are those that should be used regardless of the device configuration or when there are no alternative resources that match the current configuration.

Alternative resources

are those that you've designed for use with a specific configuration. To specify that a group of resources are for a specific configuration, append an appropriate configuration qualifier to the directory name.

Resources types

In android project, data are provided in your resources directory (res/)
Here's a brief summary of each resource type:

Animation Resources

Define pre-determined animations.

Tween animations are saved in res/anim/ and accessed from the ***R.anim*** class.

Frame animations are saved in res/drawable/ and accessed from the ***R.drawable*** class.

Color State List Resource

Define a color resources that changes based on the View state.

Saved in res/color/ and accessed from the ***R.color*** class.

Drawable Resources

Define various graphics with bitmaps or XML.

Saved in res/drawable/ and accessed from the ***R.drawable*** class.

Layout Resource

Define the layout for your application UI.

Saved in res/layout/ and accessed from the ***R.layout*** class.

Resources types - cont.

In android project, data are provided in your resources directory (res/)
Here's a brief summary of each resource type:

Menu Resource

Define the contents of your application menus.

Saved in res/menu/ and accessed from the ***R.menu*** class.

String Resources

Define strings, string arrays, and plurals (and include string formatting and styling).

Saved in res/values/ and accessed from the ***R.string, R.array***, and R.plurals classes.

Style Resource

Define the look and format for UI elements.

Saved in res/values/ and accessed from the ***R.style*** class.

More Resource Types

Define values such as booleans, integers, dimensions, colors, and other arrays.

Saved in res/values/ but each accessed from unique R sub-classes (such as ***R.bool, R.integer, R.dimen***, etc.).

Example of color state

Each color is defined in an <item> element inside a single <selector> element. Each <item> uses various attributes to describe the state in which it should be used. During each state change, the state list is traversed top to bottom and the first item that matches the current state will be used.

XML file saved at `res/color/button_text.xml` :

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:color="#ffff0000"/> <!-- pressed -->
    <item android:state_focused="true"
          android:color="#ff0000ff"/> <!-- focused -->
    <item android:color="#ff000000"/> <!-- default -->
</selector>
```

This layout XML will apply the color list to a View:

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:textColor="@color/button_text" />
```

Resources: String

- There available types:
  - String
  - String-array
  - Plural

- String can be accessed via the xml file stored in the ***res folder*** under android project.

- In xml you write***: <string name="dept">Computer Sci</string>*** now the string '***dept'*** can have deffirent values such as *Computer Sci*, or other values.

- For example: if you want to add a new language to the app then add to resources directory a file with a new language strings like values-fr. Then the previous string can be displayed as 'l'informatique'

- You can access values from Java using R.string…

Creating Value Resources

Example strings.xml

```
<resources>
    <string name="app_name">My University name</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_screen_orientation_app">
        Screen Orientation App </string>
</resources>
```

Reference in XML

- @string/app_name

Get string in code

- getString(R.string.app_name)

Creating Dimension Resources

Example dimens.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="small_size">15dp</dimen>
    <dimen name="medium_size">15sp</dimen>
    <dimen name="large_size">20pt</dimen>
</resources>
```

Reference in XML

- @dimen/small_size

Get dimension in Java code

- getResources().getDimension(R.dimen.small_size)

- dp - Density-independent pixel (dp)
- dpi - (dots per inch)
- px = dp * (dpi / 160) - 160 is the baseline density
- sp - scale-independent pixels
- pt - point 1/72 of an inch based on the physical screen

Creating Color Resources

Color formats

- #RGB, #RRGGBB, #ARGB, #AARRGGBB

Example colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="red_color">#F00</color>
    <color name="green_color">#00FF00</color>
    <color name="blue_alpha_color">#500000FF</color>
</resources>
```

Reference in XML

- @color/red_color

Get color in Kotlin code

- resources.getColor (R.color.red_color)

Creating Style Resources

Example styles.xml

```
<resources>
    <style name="AppTheme" parent="android:Theme.Light" />
    <style name="style1">
        <item name="android:textColor">#00FF00 </item>
        <item name="android:typeface">serif</item>
        <item name="android:textSize">30sp </item>
     </style>
     <style name="style2" parent="style1" >
        <item name="android:textColor">#0000FF</item>
        <item name="android:typeface">sans</item>
        <item name="android:background">#FF0000</item>
        <item name="android:padding">10dip</item>
     </style>
     <style name="style3" parent="style2" >
        <item name="android:textColor">#00FF00</item>
        <item name="android:background">#00000000</item>
        <item name="android:typeface">monospace</item>
        <item name="android:gravity">center</item>
     </style>
</resources>
```

Creating Style Resources – cont.

Add to a view in XML

- style="@style/style1"

Add to activity in Android manifest

- android:theme="@style/AppTheme"

Arrays (Strings)

A collection of values or elements

Example strings.xml

```
<resources>
    <string name="app_name">StringArrayApp</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_string_array_app">StringArrayAppActivity</string>
    <string-array name="Fruits">
        <item>Apple</item>
        <item>Mango</item>
        <item>Orange</item>
        <item>Grapes</item>
        <item>Banana</item>
    </string-array>
</resources>
```

Arrays (Strings) – Demo Example

```
package com.androidunleashed.stringarrayapp;

class MainActivity : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_string_array_app);
  val fruitsView = findViewById<View>(R.id.fruits_view) as TextView
val fruitsArray = resources.getStringArray(R.array.Fruits)
var str = ""
for (i in fruitsArray.indices) {
   str += """
      ${fruitsArray[i]}
      """.trimIndent()
}
fruitsView.text = str
       }
}
```

Arrays (Strings) – cont.

Activity_string_array_app.xml

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:id="@+id/fruits_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Drawable Resources

A drawable resource is a general concept for a graphic that can be drawn to the screen and which you can retrieve with APIs such as ***getDrawable(int)*** or apply to another XML resource with attributes such as ***android:drawable*** and ***android:icon***.

PNG, JPG, and GIF

320dpi, 240dpi, 160dpi, 120dpi (dots per inch)

drawable-xhdpi, -hdpi, -mdpi, -ldpi

How to use:

- @drawable/image_filename
- R.drawable.image_filename

http://developer.android.com/guide/topics/resources/drawable-resource.html

Drawable Resources – cont.

Shape Drawable

This is a generic shape defined in XML.
FILE LOCATION:
res/drawable/filename.xml
RESOURCE REFERENCE:
In Kotlin: R.drawable.filename
In XML: @[package:]drawable/filename

Imagine that you want to include a gradient background color for a TextView of a layout. The answer in the next example:

Drawable Resources – example 1/2

EXAMPLE:

XML file saved at `res/drawable/gradient_box.xml` :

```xml
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="#FFFF0000"
        android:endColor="#80FF00FF"
        android:angle="45"/>
    <padding android:left="7dp"
        android:top="7dp"
        android:right="7dp"
        android:bottom="7dp" />
    <corners android:radius="8dp" />
</shape>
```

Drawable Resources – example2/2

This layout XML applies the shape drawable to a View:

```
<TextView
    android:background="@drawable/gradient_box"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content" />
```

This application code gets the shape drawable and applies it to a View:

```
Resources res =  getResources() ;
Drawable shape = res.  getDrawable (R.drawable.gradient_box);

TextView tv = (TextView)findViewByID(R.id.textview);
tv.setBackground(shape);
```

Android App structure

A layout defines the visual structure for a user interface.

Using Android's XML vocabulary, you can quickly design UI layouts and the screen elements they contain, in the same way you create web pages in HTML — with a series of nested elements.

The advantage to declaring your UI in XML is that it enables you to better separate the presentation of your application from the code that controls its behavior.

Layouts host Android apps' components.
What are Android UI components?

Basic UI Components/controls

TextView – a read-only text label

EditText – an editable text box

ListView – a ViewGroup that creates and manages a vertical list of views

Spinner – a TextView and an associated list of items to allow you to select an item from the list

Button – a standard command button

CheckBox – a button allowing a user to select/check or unselect/uncheck

RadioButton – a mutually exclusive button selecting an item from a group

EditText

layout_height – e.g. wrap_content, match_parent

singleLine – true, remains on single line; false, doesn't

hint – background hint

lines – number of lines

textSize – size of text

autoText – auto correct text

capitalize – auto cap (none, characters, words, sentences)

password – hide text

minWidth – specify minimum width

maxWidth – specify max width

minHeight – specify min height

maxHeight – specify max height

scrollHorizontally – if true, scrolls to end of typed text

Toast

A toast is a view containing a quick little message for the user. The toast class helps you create and show those.

```
import android.widget.Toast;
 Toast.makeText(this, “Your Message”,
        Toast.LENGTH_SHORT).show();
```

Log

android.util.Log class used for sending log output

Generally, use:

- Log.v() Log.d() Log.i() Log.w() and Log.e() methods.

Order in terms of verbosity, from least to most is:

- ERROR, WARN, INFO, DEBUG, VERBOSE
- Verbose should never be compiled into an application except during development.

Debug logs are compiled in but stripped at runtime

Error, warning and info logs are always kept

Event Handling

Via anonymous inner class

Implement OnClickListener Interface

Declare event handler in XML control definition

Lambda

ImageView

android:src – the source

- Example:
- Var imgView = findViewById<View>(…) as ImageView
- imgView.setBackground(R.drawable.fileName)

ToggleButton

Example (setup in layout xml file)

- android:textOn="Play"
- android:textOff="Stop"

In code, setOnClickListener (anonymous class)

ScrollView

Special control set up a vertical scrollbar

Can only have ***one*** child

android:fillViewport attribute

Thank you