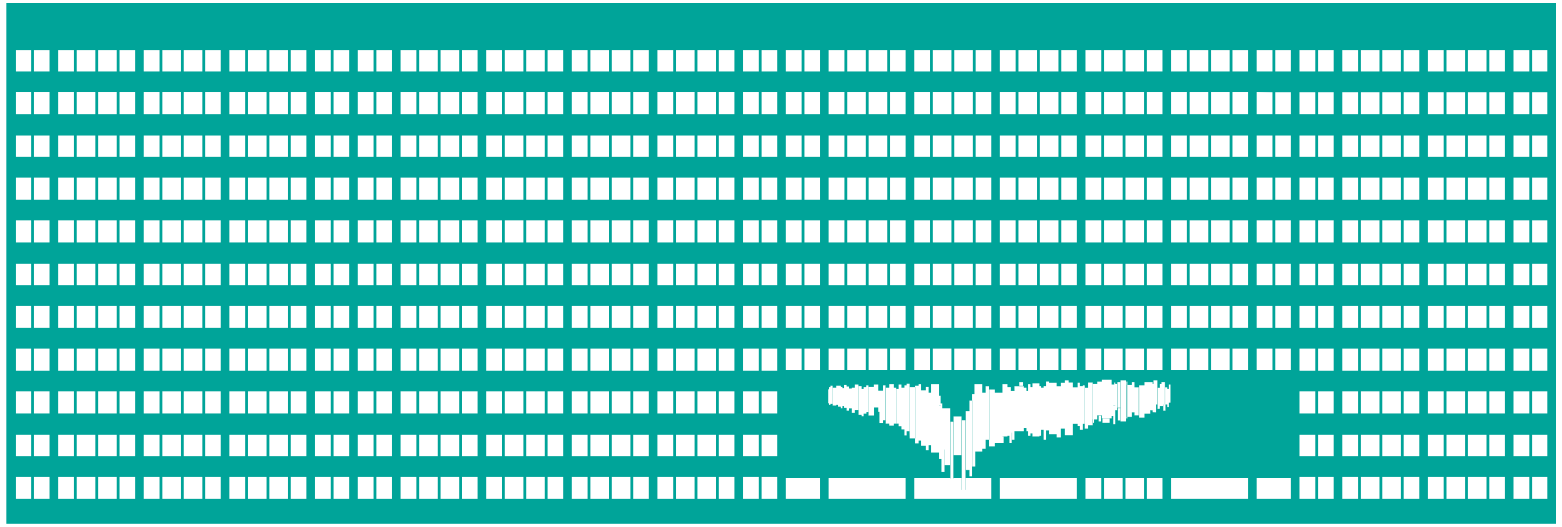
VŠB TECHNICKÁ UNIVERZITA OSTRAVA

VSB TECHNICAL UNIVERSITY OF OSTRAVA



www.vsb.cz

VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

Android – Application Internals

Michal Krumnikl

Types of Applications

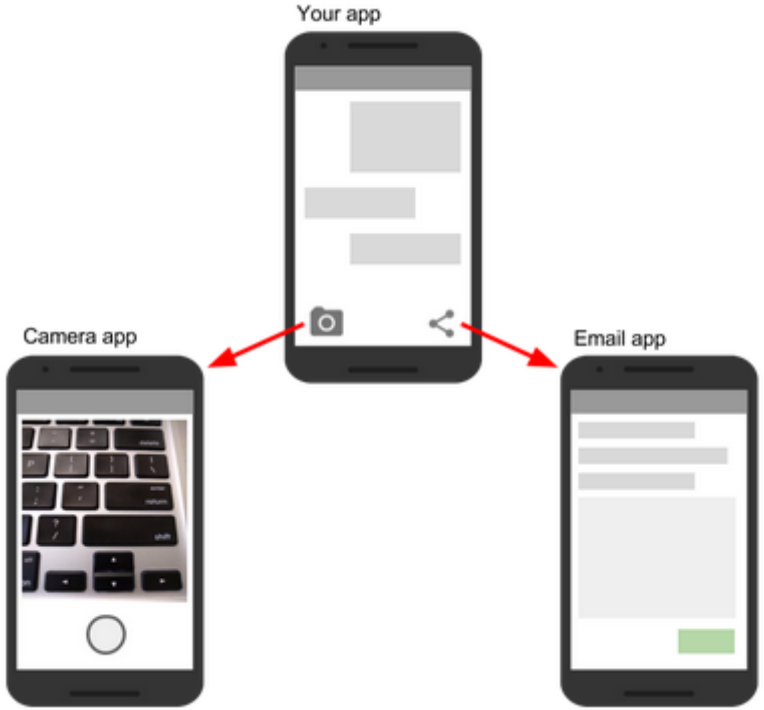
- **Foreground**
  - Useful only in foreground, suspended when it's not visible.
- **Background**
  - Limited interaction, usually only configuration. Spend most of time hidden, e.g. call screening application, automatic SMS responder etc.
- **Intermittent**
  - Expects some interactivity, but does the most of its work in the background, e.g. media player.
- **Widget**
  - Application visible only as a home screen widget, e.g. battery indicator, weather forecast.

Application Components

- **Activities** - ***android.app.Activity***
  - Primary class for user interaction
  - Analogue to window or dialog boxes on desktop
  - Application's presentation layer
  - An activity is implemented as a subclass of Activity
- **Services** - ***android.app.Service***
  - Code designed to be kept running on the background
  - Independent of any activitity
  - A service is implemented as a subclass of Service
- **Content Providers** - ***android.content.ContentProvider***
  - Store & share data across applications
  - Data can be accessible by multiple applications
  - Implemented as a subclass of ContentProvider and must implement a standard set of APIs
- **Broadcast Receivers** - ***android.content.BroadcastReceiver***
  - The subscriber in publish/subscribe pattern
  - Responds to system-wide broadcast announcements.
  - May create a status bar notification
  - Can even automatically start your application to respond to an incoming Intent, creating event-driven applications.
- **Widgets** - ***android.appwidget***
  - Visual components added to the home screen.
  - Notifications
  - Notifications can signal user information without stealing focus or interrupting current Activities
  - Preferred technique for getting user attention from within a Service or Broadcast Receiver

Activities

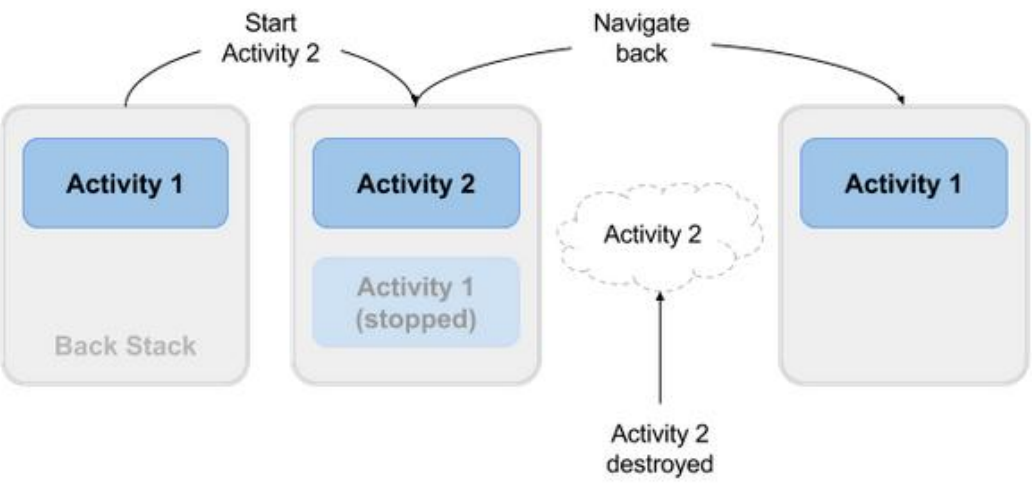
- **Application can define one or more activities** to handle different phase of the program.
- **Provides a screen** with which users can interact.
- Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack.
- Activity is responsible **for saving its own state** so it can be restored later.
- Activity can be in several states. Application gets notified when the state is going to change through the *onXXX()* method calls.
- You can override these methods in Activity class.



Different „Implementations“

- ***Activity***
  - **Base Class**: Activity is the base class for activities in Android. Provides the core functionality required to create an activity.
  - **No Support Library Features**: It does not include any of the features provided by the Android Support Library, which means it may not support newer UI or features introduced in later versions of Android.
- ***AppCompatActivity***
  - **Support Library**: AppCompatActivity is a subclass of Activity provided by the AndroidX (previously the Android Support Library). It is designed to provide backward compatibility for modern Android features on older devices.
  - **Material Design Support**: It offers support for Material Design components and themes, which makes it easier to create a consistent user interface across different Android versions.
  - **Action Bar Support**: It includes built-in support for the Action Bar.
  - **Lifecycles and Fragments**: It provides additional lifecycle methods and fragment management.
- ***ComponentActivity***
  - Kotlin only base class for activities that enables composition of higher level components.
  - Contains everything for a **Compose-only app.**

Activity Lifecycle



- **Most important callback methods**
  - **onCreate(Bundle savedInstanceStat)**
    - Must be implemented. The system calls this when creating an activity. Within your implementation, you should initialize the essential components of your activity. Most importantly, this is where you must call setContentView() to define the layout for the activity's user interface.
  - **onPause()**
    - The system calls this method as the first indication that the user is leaving your activity (though it does not always mean the activity is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back).

Activity Lifecycle

protected void onStart()

- Activity is about to become visible

protected void onResume()

- Activity is visible and about to start interacting with user

protected void onRestart()

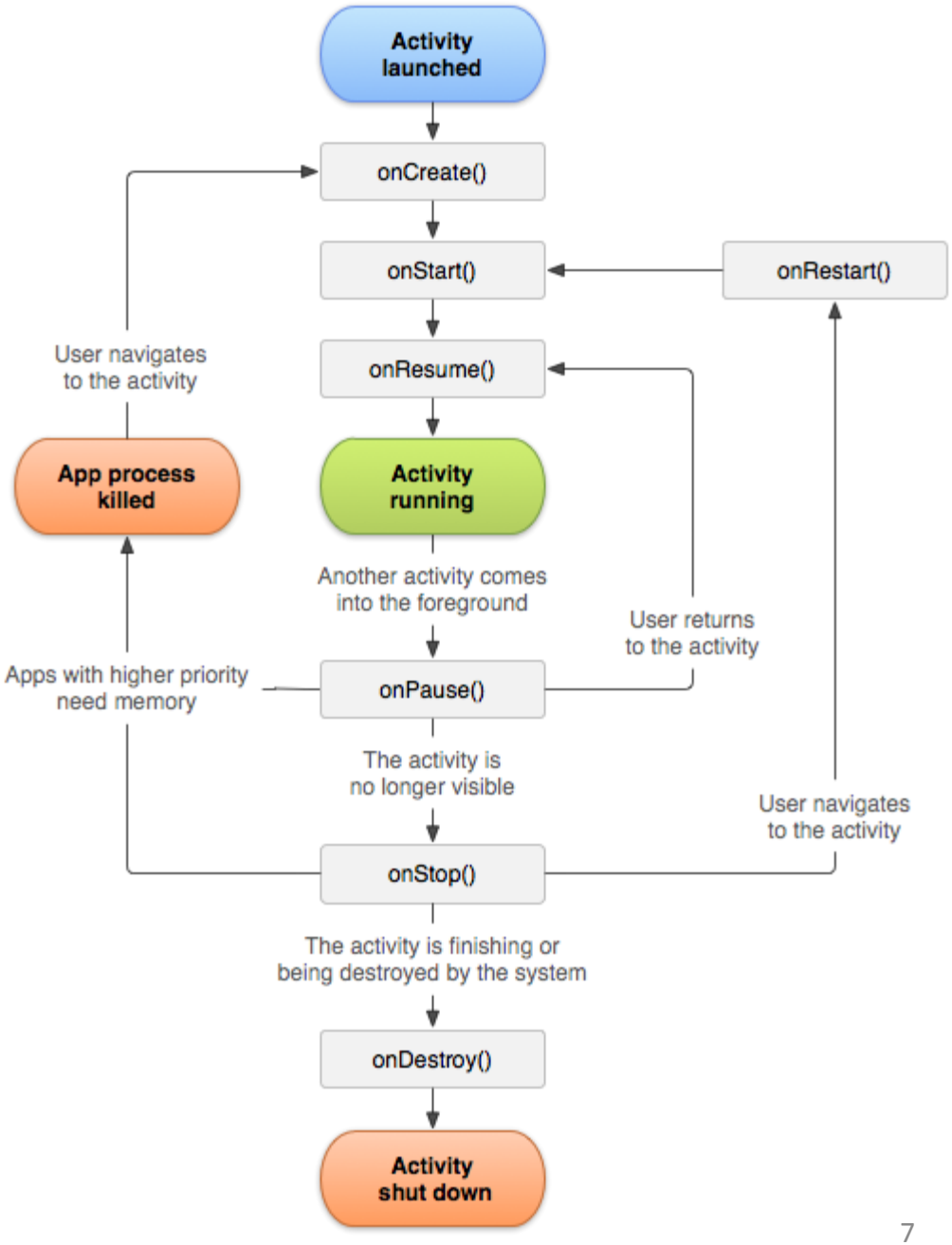
- Called if the Activity has been stopped and is about to be started again

protected void onStop()

- Activity is no longer visible to user. May be restarted later

protected void onDestroy()

- Activity is about to be destroyed



Activity Lifecycle

protected void onStart()

- Activity is about to become visible

protected void onResume()

- Activity is visible and about to start interacting with user

protected void onRestart()

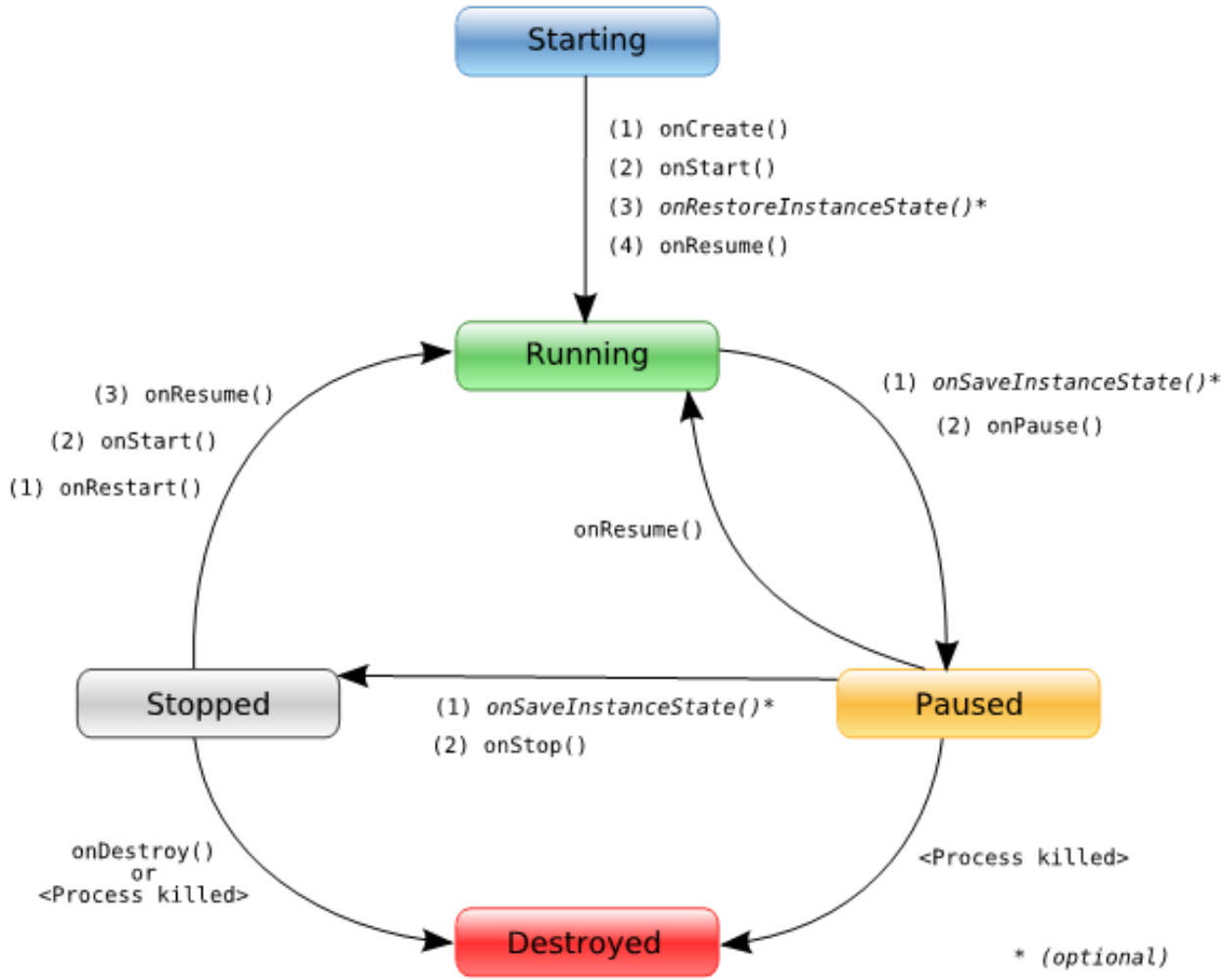
- Called if the Activity has been stopped and is about to be started again

protected void onStop()

- Activity is no longer visible to user. May be restarted later

protected void onDestroy()

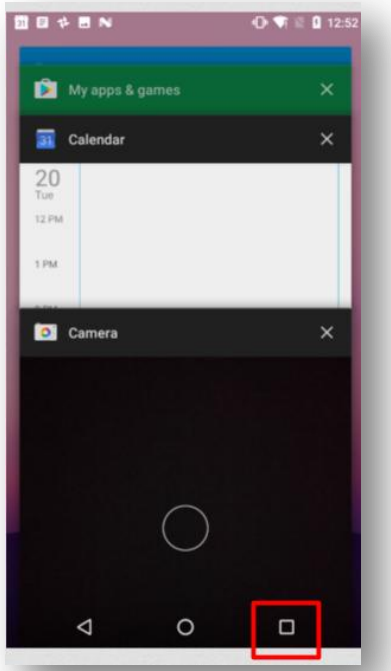
- Activity is about to be destroyed



Navigation Through Activities

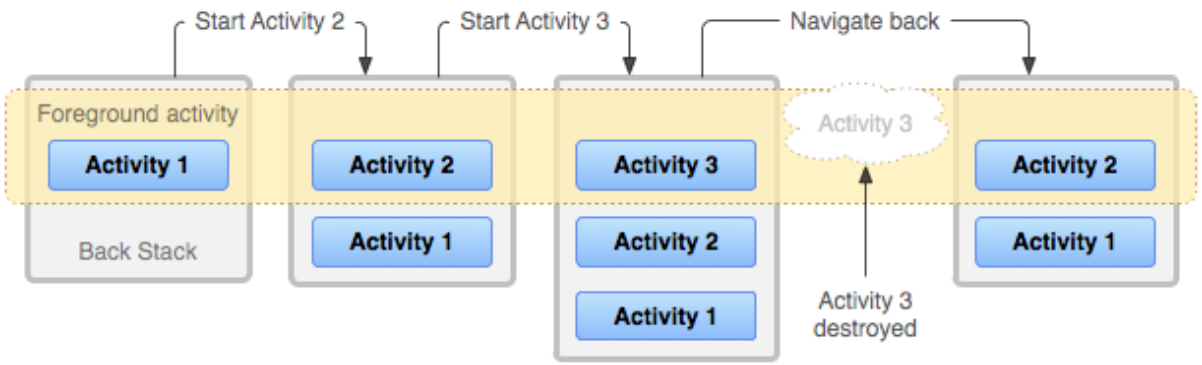
Android supports navigation in several ways:

- **Tasks**
  - A set of related Activities
  - These related activities don't have to be part of the same application
  - Most tasks start at the home screen
- **Backstack**
  - When an Activity is launched, it goes on top of the backstack
  - When the Activity is destroyed, it is popped off.
- **Suspending & resuming activities**
  - Some of these actions depend on user behavior
  - Some depend on Android
    - e.g., Android can kill activities when it needs their resources



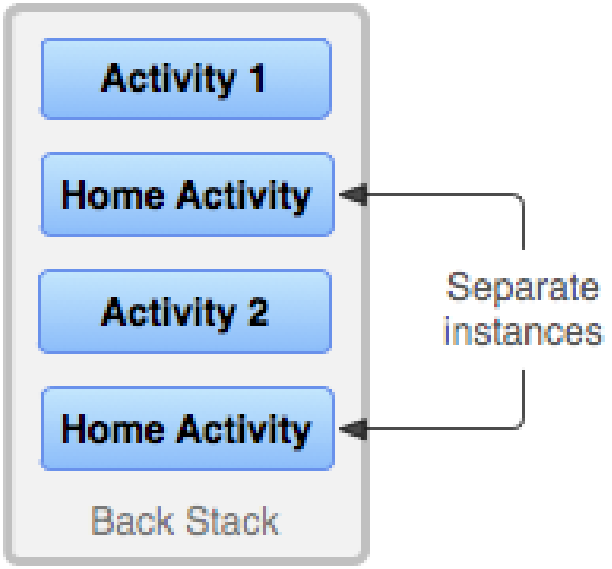
Tasks and Back Stack

- Application usually contains multiple activities.
- A **task** is a collection of activities that users interact with when performing a certain job. The activities are arranged in a stack (the "**back stack**"), in the order in which each activity is opened.
  - Back stack operates as a "last in, first out"



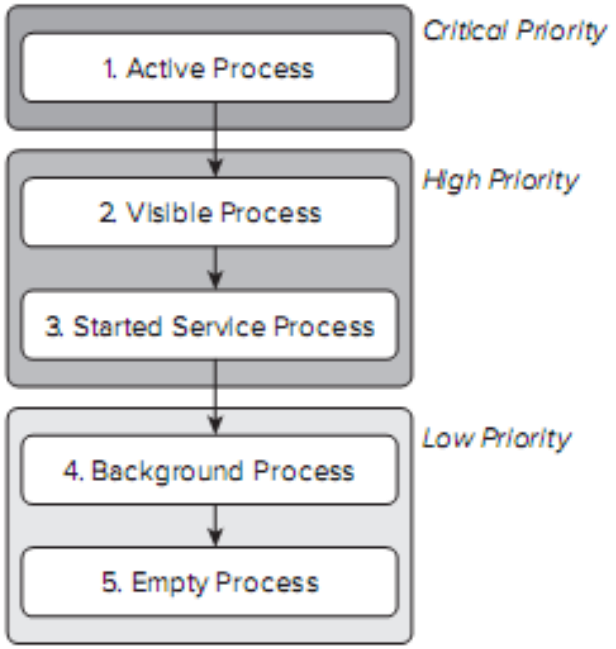
Tasks and Back Stack

- Because the activities in the **back stack are never rearranged**, if your application allows users to start a particular activity from more than one activity, a new instance of that activity is created and pushed onto the stack (rather than bringing any previous instance of the activity to the top).



Application Priorities

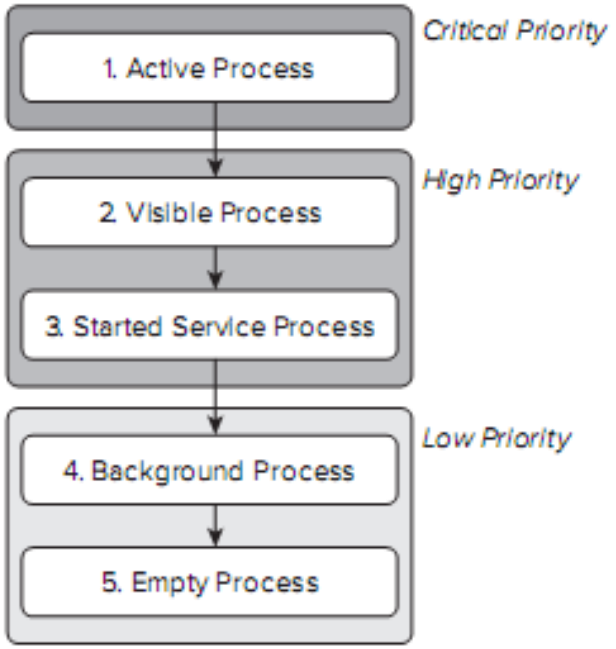
- All Android applications **will remain running** and in memory until the system needs resources for other applications.
- The order in which processes are killed to reclaim resources is determined by the priority of the hosted applications. An application's priority is equal to its highest-priority component.
- If two applications have the same priority, the process that has been at a lower priority longest will be killed first. Process priority is also affected by interprocess dependencies



https://developer.android.com/reference/android/app/Activity#process-lifecycle

Application Priorities

- **Active processes**
  - Foreground processes interacting with the user
- **Visible processes**
  - Visible but inactive. They are not in foreground.
- **Started Service processes**
  - Processing that should continue without a visible interface.
- **Background processes**
  - Processes without running services.
- **Empty processes**
  - Retain an application after it has reached end in order to improve system performance.



Implementing Activities

- Activity initialization code usually in ***onCreate()***
- Typical ***onCreate()*** workflow:
  - Restore saved state
  - Set content view
  - Initialize UI elements
  - Link UI elements to code actions

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        …
    }
```

```
class MainActivity : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      enableEdgeToEdge()
      setContentView(R.layout.activity_main)
      ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) {
       v, insets ->
          val systemBars =
       insets.getInsets(WindowInsetsCompat.Type.systemBars())
          v.setPadding(systemBars.left, systemBars.top, systemBars.right,
       systemBars.bottom)
          insets
      }
  }
}
```

Starting Activities

- Create an **Intent** object specifying the Activity to start
- Pass newly created Intent to methods, such as:
  - **startActivity()**
  - **startActivityForResult()**

- Invokes a Callback method when the called Activity finishes to return a result.
- Started Activity can set its result by calling
  - **Activity.setResult()**

Configuration Changes

- Device configuration can change at runtime
  - Keyboard, orientation, locale, etc.
- **On configuration changes, Android usually kills the current Activity and then restarts it**
- If necessary you can:
  - **Retain an Object** containing important state information during a configuration change
  - **Manually handle the configuration change**
- Android **automatically saves the states of Views with unique ID**. Additionally, it can be manually handled by
  - protected void **onSaveInstanceState**(Bundle outState);
  - protected void **onRestoreInstanceState**(Bundle savedInstanceState);

Instance state

- If the system **destroys the activity due to system constraints** (such as a configuration change or memory pressure), then although the actual Activity instance is gone, the system remembers that it existed.
  - *onSaveInstanceState()* is not called when the user explicitly closes the activity or in other cases when finish() is called.

```
@Override
public void onSaveInstanceState(Bundle
       savedInstanceState) {
    // Save the user's current game state.
    savedInstanceState.putInt(STATE_SCORE, currentScore);
    savedInstanceState.putInt(STATE_LEVEL, currentLevel);

    // Save the view hierarchy state.
    super.onSaveInstanceState(savedInstanceState);
}
```

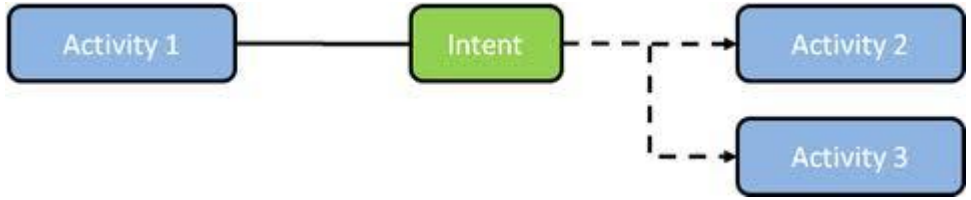
```
public void onRestoreInstanceState(Bundle savedInstanceState) {
    // Restore the view hierarchy.
    super.onRestoreInstanceState(savedInstanceState);

    // Restore state members from saved instance.
    currentScore = savedInstanceState.getInt(STATE_SCORE);
    currentLevel = savedInstanceState.getInt(STATE_LEVEL);
}
```

Intents

- A mechanism for **describing a specific action**, e.g. “send a message”, “phone to office”.
  - There is an intent for “send an email”. If your application needs this you can invoke this event. If you’re writing new email application, you can register your activity to handle this intent and replace the standard program.
- Facility for performing **late runtime binding between the code in different applications**. It is basically a passive data structure holding an abstract description of an action to be performed.
- An Intent object is a **bundle of information**. It contains information of interest to the component that receives the intent

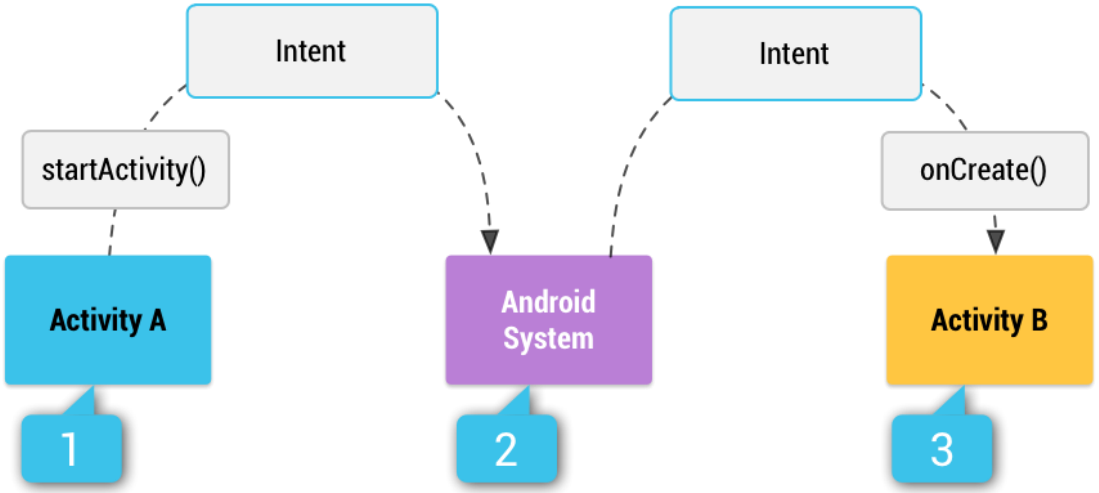
Intents

- In Android everything goes through intents.
- A passive data structure holding an abstract description of an operation to be performed
- Intents provide a flexible language for specifying operations to be performed
- More practically, it represents
  - An **operation** to be performed
  - An **event** that has occurred
- Used to send asynchronous messages
  - Send or receive data from and to other activities or services
  - Communicate between any installed application component



Intents

- **Explicit intents**
  - Names the component, e.g. the class which should be called.
- **Implicit intents**
  - Asks the system to perform a service without telling the system which Java class should do this service.



Intents Objects

- **Component name** (optional)
  - The name of the component that should handle the intent.
  - e.g., com.example.project.app.SomeActivity
- **Action**
  - A string **naming the action** to be performed.
  - You can also define your own action strings for activating the components in your application.

Constant	Target component	Action
ACTION_CALL	activity	Initiate a phone call.
ACTION_EDIT	activity	Display data for the user to edit.
ACTION_MAIN	activity	Start up as the initial activity of a task, with no data input and no returned output.
ACTION_SYNC	activity	Synchronize data on a server with data on the mobile device.

Intents Objects

- **Data**
  - The URI of the **data to be acted on** and the MIME type of that data. Different actions are paired with different kinds of data specifications.
  - E.g., if the action field is ACTION_EDIT, the data field would contain the URI of the document to be displayed for editing. If the action is ACTION_CALL, the data field would be a tel: URI with number.
- **Category**
  - A string containing additional information about the **kind of component** that should handle the intent.

Constant	Meaning
CATEGORY_BROWSABLE	The target activity can be safely invoked by the browser to display data referenced by a link — for example, an image or an e-mail message.
CATEGORY_GADGET	The activity can be embedded inside of another activity that hosts gadgets.

Intents Objects

- **Extras**
  - Key-value pairs for additional information that should be delivered to the component handling the intent.
  - The Intent object has a series of *put...()* methods for inserting various types of extra data and a similar set of *get...()* methods for reading the data.
- **Flags**
  - Flags of various sorts. Many instruct the Android system how to launch an activity, e.g.
    - https://developer.android.com/reference/android/content/Intent#flags
  - **FLAG_ACTIVITY_SINGLE_TOP**
    - If set, the activity will not be launched if it is already running at the top of the history stack.
  - FLAG_ACTIVITY_NO_HISTORY
    - Don’t put this Activity in the History stack
  - FLAG_DEBUG_LOG_RESOLUTION
    - Print extra logging information when this Intent is processed

Implicit Activation

- When the Activity to be activated is not explicitly named, Android tries to find Activities that match the Intent.
- This process is called **intent resolution.**

- **Intent resolution process** is driven by
  - **An Intent** describing a desired operation
  - **Intent Filters** which describe which operations an Activity can handle
    - Specified either in AndroidManifest.xml or programmatically

Intent Filter

- Determine suitable applications for an implicit intent.
- If several applications exists offer the user the choice.
- The determination is based on **intent filters**.
- Intent filters are defined via the *AndroidManifest.xml*

To investigate intent filters run:
adb shell dumpsys package

- To react to a certain implicit intent an application component must register itself via an **IntentFilter** in the *AndroidManifest.xml* to this event.
- If a component **does not define intent filters** it can only be called by **explicit intents**.

Intent Filter

- Resolution data
  - **Action** (intent filter can declare zero or more <action> elements)
  - **Data** (both URI & TYPE)
  - **Category**

```
<activity ...>
        <intent-filter ...>
         <action android:name="android.intent.action.EDIT" />
         <action android:name="android.intent.action.VIEW" />
         <category android:name="android.intent.category.DEFAULT" />
         <category android:name="android.intent.category.BROWSABLE" />
         <data android:scheme="http" android:mimeType="video/*" />
        </intent-filter>
...
</activity>
```

Intents and Activities

- An activity is started using the method ***startActivity(Intent)*** if you do not need a return value from the called activity.

```
intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://tamz2.mrl.cz"));
startActivity(intent);
```

- Use the method ***startActivityForResult()*** if you need a return value. Once the called Activity is finished the method ***onActivityResult()*** in the calling activity will be called.
  - If you use ***startActivityForResult()*** then the activity which is started is considered a "Sub-Activity".

```
intent = new Intent("android.media.action.IMAGE_CAPTURE");
startActivityForResult(intent, INT_CODE);
    …
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode == Activity.RESULT_OK && requestCode == 0) {
        ….
    } }
```

Example of Implicit Intent & Filter

If you want your activity to receive implicit intents, it must include "android.intent.category.DEFAULT" in its intent filters.

in AndroidManifest.XML

```
<activity android:name=".ExampleActivity"
android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
</activity>
```

in Code

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.setType("text/plain");
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
// Start the activity
startActivity(sendIntent);
```

Finding if Intent Exists

- You can find if an application is available for a certain intent by checking the *PackageManager*. The following code checks in runtime if an intent exists.

```
public boolean isIntentAvailable(Context context, String action) {
    final PackageManager packageManager = context.getPackageManager();
    final Intent intent = new Intent(action);
    List<ResolveInfo> resolveInfo = packageManager.queryIntentActivities(intent,
                PackageManager.MATCH_DEFAULT_ONLY);

   if (resolveInfo.size() > 0) {
            return true;
   }
   return false;
}
```

Finding if Intent Exists

- You can find if an application is available for a certain intent by checking the *PackageManager*. The following code checks in runtime if an intent exists.

```
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(chooser);
}
```

To start activity from CLI using ADB

adb shell am start -n com.example.demo/com.example.test.MainActivity

Application Manifest

- Manifest defines the **Application structure**, its metadata, components and requirements
- Nodes for each components (Activities, Services, Content Providers and Broadcast receivers)
- Also defines **intent filters** and **permissions.**
- Used for security settings, unit tests and also defines hardware requirements

```
<manifest xmlns:android=http://schemas.android.com/apk/res/android
     package="com.my_domain.my_app"
     android:versionCode="1"
     android:versionName="0.9 Beta">
                [ ... manifest nodes ... ]
</manifest>
```

Application Manifest

- **`uses-sdk`** node define a minimum, maximum and target SDK version that **must be available** on a device or emulator in order for your application to work.
- It is used on Market to filter the application
- Supported SDK version is not equivalent to platform version. (Android 1.6 – API v4, 2.2 – API v8 …)
  - Minimum SDK – lowest version of the SDK that includes the APIs you've used in your application
  - Maximum SDK – upper limit you are willing to support.
  - Target SDK – platform used for development

```
<uses-sdk android:minSdkVersion="4"
     android:targetSdkVersion="5">
</uses-sdk>
```

Application Manifest

- **`uses-configuration`** node specifies combination of **supported input devices**.
  - reqFiveWayNav, reqHardKeyboard, reqNavigation, reqTouchScreen …
- You can specify multiple supported configurations, but be aware that the application will not be installed on an device that does not have one of the specified combination.
- Usually this node is not necessary, since the application should work with any input configurations.

Application Manifest

- **`uses-feature`** node specifies the necessary **hardware features** application requires.
- This way you can prevent to install application that would become useless without specific hardware
  - Android.hardware.camera,
  - Android.hardware.camera.autofocus.
  - android.hardware.touchscreen.multitouch
  - Android.hardware.telephony.cdma, etc.
- It is also used for specifying minimum version of **OpenGL**

```
<uses-feature android:glEsVersion=" 0x00010001"
android:name="android.hardware.camera" />
```

Application Manifest

- **`supports-screens`** node specifies the screen sizes your application can, and can't, support.
  - **smallScreens** - Screens with a resolution smaller than traditional HVGA—typically QVGA screens.
  - **normalScreens** - Used to specify typical mobile phone screens of at least HVGA, including WVGA and WQVGA.
  - **largeScreens** - Screens larger than normal. In this instance a large screen is considered to be significantly larger than a mobile phone display.
  - **anyDensity** Set to true if your application can be scaled to accommodate any screen resolution.

```
<supports-screens   android:smallScreens=["false"]
                    android:normalScreens=["true"]
                    android:largeScreens=["true"]
                    android:anyDensity=["false"] />
```

Application Manifest

- Single **`application`** node specifies the metadata of you application (title, icon, theme …)
- Also contains ***debuggable* attribute**, that will provide an additional information for developers.
- Also acts as a container including Activities, Services, Content Providers and other components.

```
<application android:icon="@drawable/icon"
      android:theme="@style/my_theme"
      android:name="MyApplication"
      android:debuggable="true">
      [ ... application nodes ... ]
</application>
```

Application Manifest

- **`Activity`** nodes are required for every Activity displayed by your application.
- **You must include all activities** including the main launch activity and any other screen or dialog that can be displayed.
- A runtime exception is raised when an attempt to start a non-defined activity occurs.
- Services, provider, receiver are defined similarly.

```
<activity android:name=".MyActivity" 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

- Home application populates the app launcher by finding all the activities with intent filters that specify the **ACTION_MAIN** action and **CATEGORY_LAUNCHER** category.
- An intent filter is registered which defines that **this activity is started** once the application starts

```
action android:name="android.intent.action.MAIN
```

- The category definition

```
category android:name="android.intent.category.LAUNCHER
```

defines that this application is added to **the application directory** on the Android device.

Application Manifest

- There are four launch (`android:launchMode`) modes available as part of the `activity`
- `standard` (the default) - A new Activity is launched and added to the back stack for the current task. An Activity can be instantiated multiple times, a single task can have multiple instances of the same Activity, and multiple instances can belong to different tasks.
- `singleTop` - If an instance of an Activity exists at the top of the back stack for the current task and an Intent request for that Activity arrives, Android routes that Intent to the existing Activity instance rather than creating a new instance. A new Activity is still instantiated if there is an existing Activity anywhere in the back stack other than the top.
- `singleTask` - When the Activity is launched the system creates a new task for that Activity. If another task already exists with an instance of that Activity, the system routes the Intent to that Activity instead.
- `singleInstance` - Same as single task, except that the system doesn't launch any other Activity into the task holding the Activity instance. The Activity is always the single and only member of its task.

Application Manifest

- Permissions are required for many of the native Android services, particularly those with a cost or security implication (such as dialing, receiving SMS, or using the location-based services).
- Permission failure will result in a *SecurityException*

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

- Other security concerns
  https://developer.android.com/privacy-and-security/risks/android-debuggable

Resources

- **Resources are non-source code entities, additional files and static content that your code uses.**
- Many different resource types
  - Layout,
  - Strings,
  - Images,
  - Menus
  - Animations
- Allows applications to be customized for different devices and users
- http://developer.android.com/guide/topics/resources/overview.html

Creating Resources

- **Application resources** are located in `/res` folder
  - layout folder contains default layout, application etc.
  - drawable resources are usually created for different displays DPI and orientation
- During the built, resources are effectively compiled and included in application.

- **R class** file is automatically generated. It contains references to each included resource.
  - Java code uses the R class to access resources
- Resource file names should contain only lowercase letters, numbers, and the period and underscore symbol (. _)

Using Resources

- **Resources** can be accessed in code using the static R class. R class is generated automatically every time the application is compiled.
- Each of the subclasses within R exposes its associated resources as variables.
  - `R.string.button1_text`
- When you need an **instance of the resource** itself, you need to use helper methods.

```
• Resources myRes = getResources();
• Drawable icon = myRes.getDrawable(R.drawable.app_icon);
```

- **Resources within Resources**
  - Use @ notation as the prefix.
  - `android:text="@string/button_label_toast"`

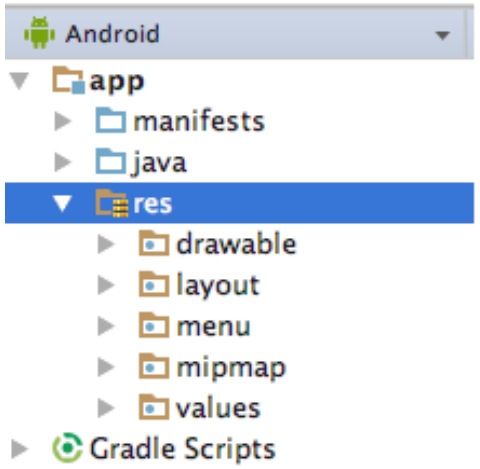
Accessing Resources

```
[<pkg_name>.]R.<resource_type>.<resource_name>
```

- **<pkg_name>** is the name of the package in which the resource is located (not required when referencing resources from your own package).
- **<resource_type>** is the R subclass for the resource type.
- **<resource_name>** is either the resource filename without the extension or the android:name attribute value in the XML element (for simple values).

- **In code :** R.string.hello
- **In XML :** @string/hello

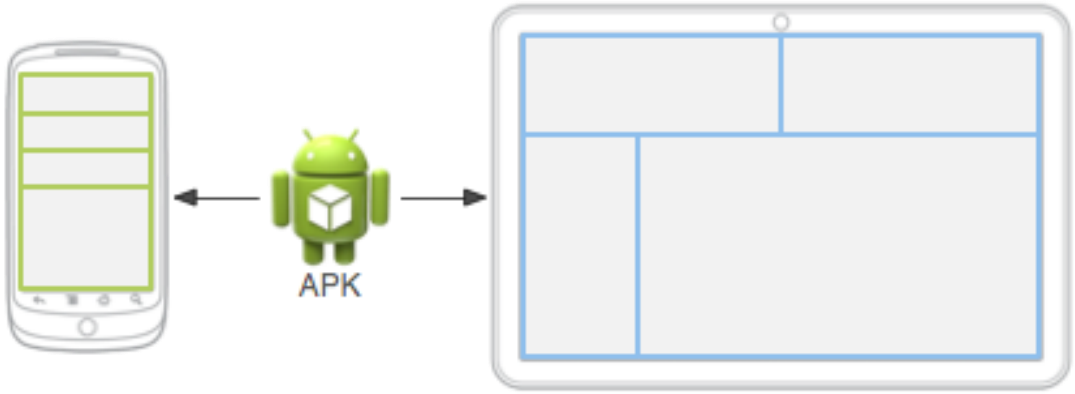
Resource Directories



Directory	Resource Type
animator/	Defining property animations
anim/	Defining tween animations
color/	Define a state list of colors
drawable/	Bitmap files (bitmaps, 9patches, shapes, …)
layout/	Define a user interface layout
menu/	Define application menus (options, context, …)
raw/	Arbitrary files save in their raw format
values/	Simple values as strings, integers, colors
xml/	Arbitrary XML files

Default and Alternative Resources

- **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.



Providing Alternative Resources

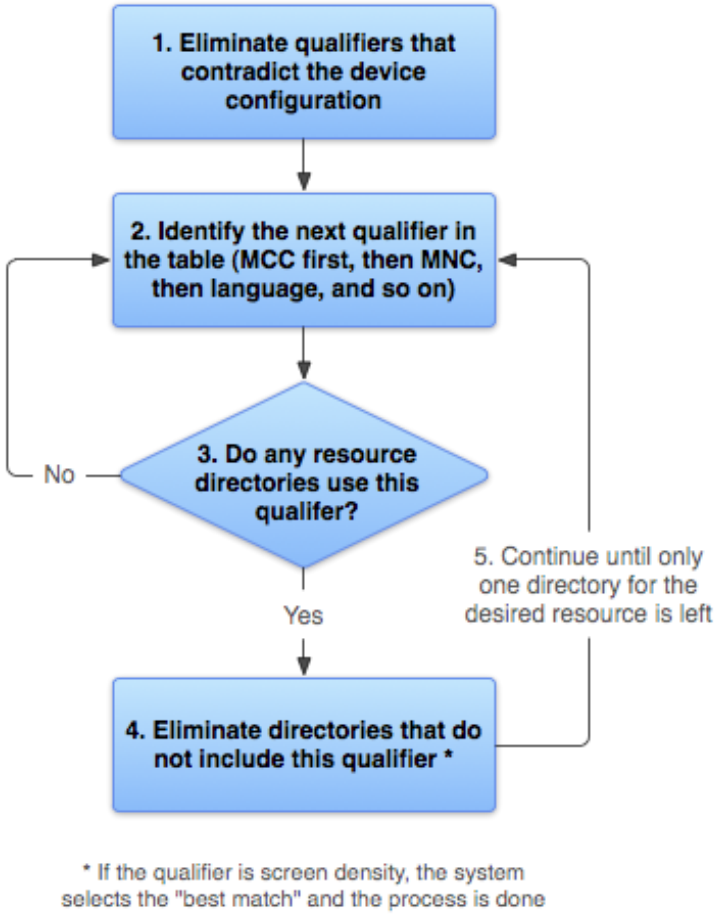
- Directory ***<resources_name>-<config_qualifier>***
  - <resources_name> is the directory name of the corresponding default resources
  - <qualifier> is a name that specifies an individual configuration for which these resources are to be used
- Configuration qualifier names
  - MCC and MNC (e.g. *mcc310-mnc004*)
  - Language and region (e.g. *en, cs, ..* see ISO 639-1)
  - Smallest width (e.g. *sw320dp, sw600dp*)
  - Available width, height (e.g. *w720dp, h720dp*)
  - Screen size (e.g. *small, large, normal*)
  - Screen orientation (*port, lang*)
  - Screen pixel density (e.g. *ldpi, mdpi, hdpi*)
- Ordering rule (e.g. *drawable-en-rUS-land*)

Selecting Best Matching Resource

- Android selects which alternative resource to use at runtime, depending on the device configuration.

Locale = **en-GB**
Screen orientation = **port**
Screen pixel density = **hdpi**
Touchscreen type = **notouch**
Primary text input method = **12key**

res/
drawable/
drawable-en/
drawable-fr-rCA/
drawable-en-port/
drawable-en-notouch-12key/
drawable-port-ldpi/
drawable-port-notouch-12key/



String and Color Resources

- **String Resources**
  - Externalizing strings helps maintain consistency within application and makes localization much more easier.
  - Types: String, String Array, Plurals
  - String resources are defined by **`<string>`** tags.
    - Android supports text styling by using HTML tags **`<b>`**, **`<i>`** and **`<u>`**
- **Color Resources**
  - Color values are specified using # symbol followed by the (optional) alpha channel, and RGB values.
  - Following notations are possible:
    - #RGB
    - #RRGGBB
    - #ARGB
    - #AARRGGBB

Style and Themes Resources

- **Style Resources** can be used to maintain a consistent look and feel of the application.
- The specify the attribute values used by Views. The most common use is to store colors and fonts.
- To create a style **`<style>`** tag is used that includes a **`name`** attribute and contains one or more **`item`** tags.
- Each **`item`** tag include attribute to specify the attribute such as font, color etc.

Layout Resources

- **Layout resources** provides mechanism to create interface layouts in XML rather than constructing them in code.
- Layouts are usually used for defining user interface for Activity. Once defined, they are "inflated" within an Activity using **`setContentView`**, usually within **`onCreate`** method.
- Each layour definition is stored in a separate file, each containing a single layour in **`res/layout`**.
- Using layouts is best-practice UI design.

Other Resources

- **Dimensions** are most commonly referenced within style and layout resources.
- **Drawable resources** include bitmaps and NinePatch (stretchable PNG) images.
- **Animations.** Android supports two types of animation. Tweened animations are used to rotate, move, stretch and fade a View. Frame-by-Frame animations let you create a sequence of Drawables.
- **Menus.** Menu layouts can be also saved in XML. Each menu is stored in separate file.

Assets

- While the directory "res" contains structured values which are known to the Android platform the directory "assets" can be used to store **any kind of data**.
- In Java you can access this data via the **AssetsManager** and the method *getAssets()*.

References

https://developer.android.com/guide/components/activities/intro-activities

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/unit-1-get-started/lesson-2-activities-and-intents/2-1-c-activities-and-intents/2-1-c-activities-and-intents.html

Android Developer Fundamentals (V2) course
https://drive.google.com/drive/folders/1eu-LXxiHocSktGYpG04PfE9Xmr_pBY5P

Thank you for your attention

Mgr. Ing. Michal Krumnikl, Ph.D.

+420 597 325 867

michal.krumnikl@vsb.cz

www.vsb.cz