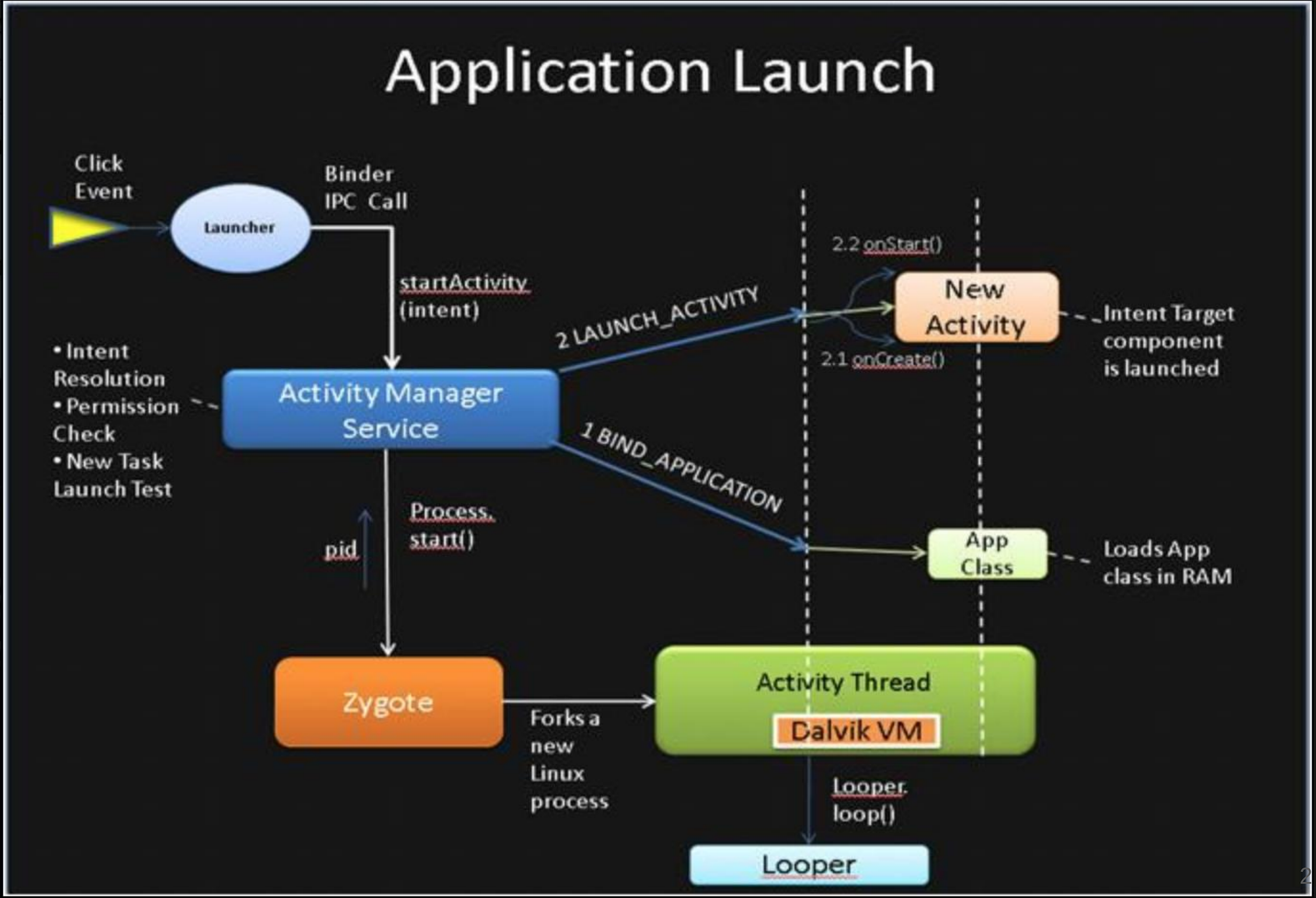
Ganpat University
॥ विद्यया समाजोत्कर्षः ॥
U.V. Patel College of Engineering

Mobile Application Development

U. V. Patel College of Engineering, Ganpat University of Dalvik virtual machine and pre-loads all the common classed used by the application framework



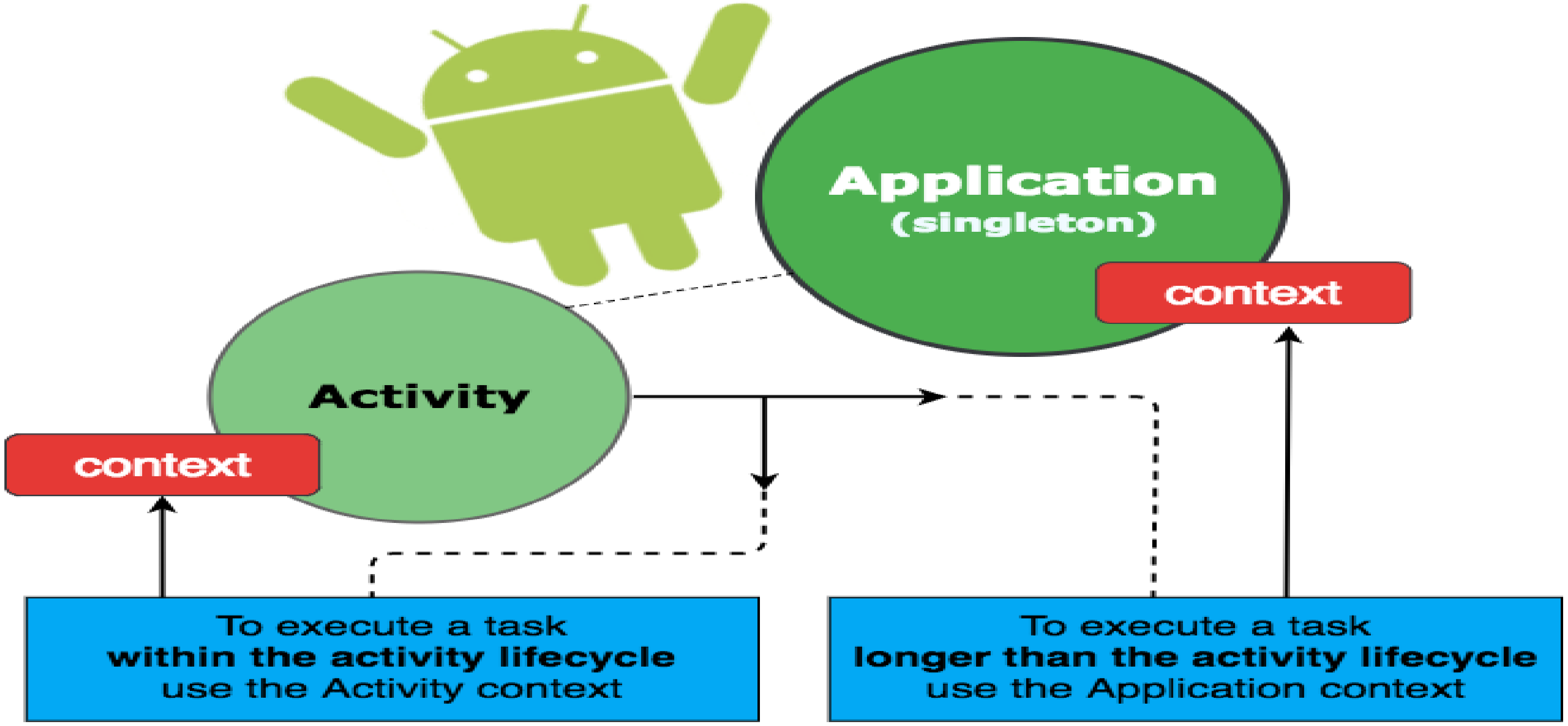
Anatomy of Android Application

- An application consists of one or more *components that are defined in the application's manifest file.*
- *A component can be one of the following:*
- ***Context***
- ***Activity***
- ***Service***
- ***Broadcast receiver***
- ***Content provider***
- ***Intent***

Context:

- **The context is the central command center for an Android application.**
- All application-specific functionality can be accessed through the context
- You can retrieve the Context for the current process using the getApplicationContext()
- Context context = getApplicationContext();

Context:



To execute a task **within the activity lifecycle** use the Activity context

To execute a task **longer than the activity lifecycle** use the Application context

Context:

Context is an abstract class, implemented in the Android system by some classes like e.g. Activity, Service, Application.

Why I need it?

Context:

- *We can use it to retrieve resources, start a new Activity, show a dialog, start a system service, create a view, and more.*
- *Context type depends on the Android component that lives within it.*

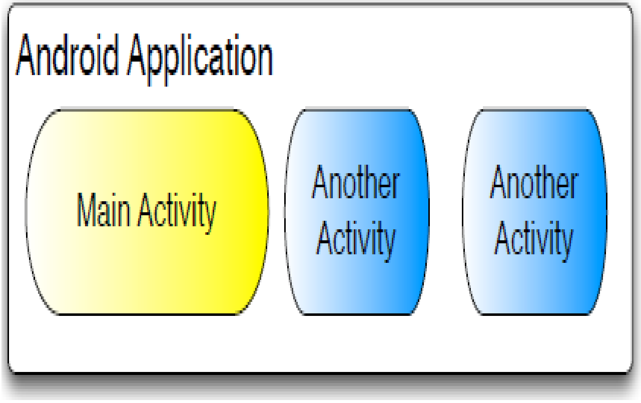
Context:

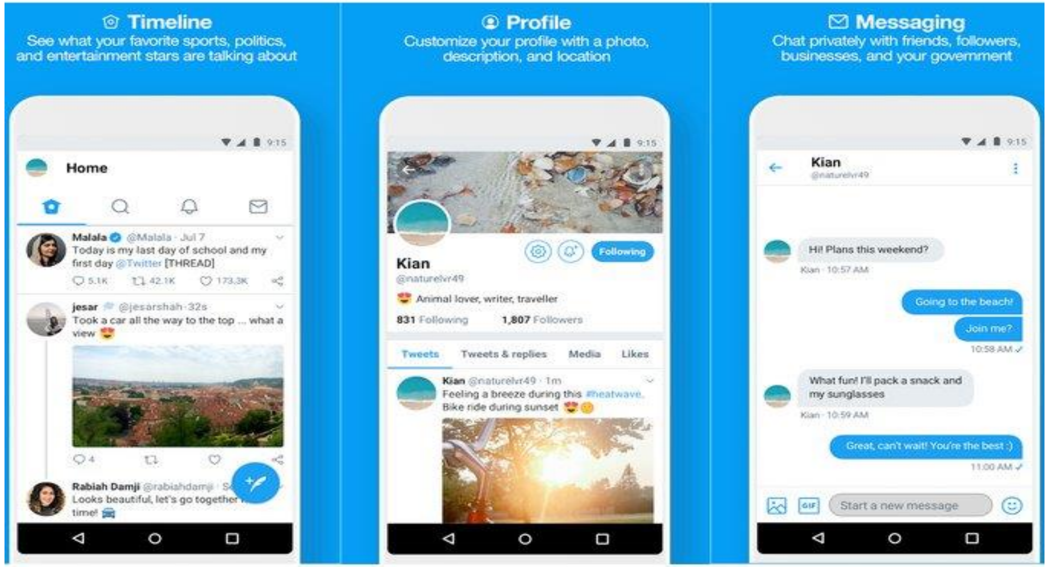
- *getContext() — returns the Context which is linked to the Activity from which is called,*
- *getApplicationContext() — returns the Context which is linked to Application which holds all activities running inside it,*

Activity:

- An Activity represents a screen or a window. Sort of.
- *each activity is independent of the others.*
- *one of the activities is marked as the first one that should be presented to the user when the application is launched.*

Activity:




Timeline
See what your favorite sports, politics, and entertainment stars are talking about

Home

Malala @Malala · Jul 7
Today is my last day of school and my first day @Twitter [THREAD]
5.1K 42.1K 173.3K

jesar @jesarshah · 32s
Took a car all the way to the top ... what a view 😍

Rabiah Damji @rabiahdamji
Looks beautiful, let's go together time! 🚘

Profile
Customize your profile with a photo, description, and location

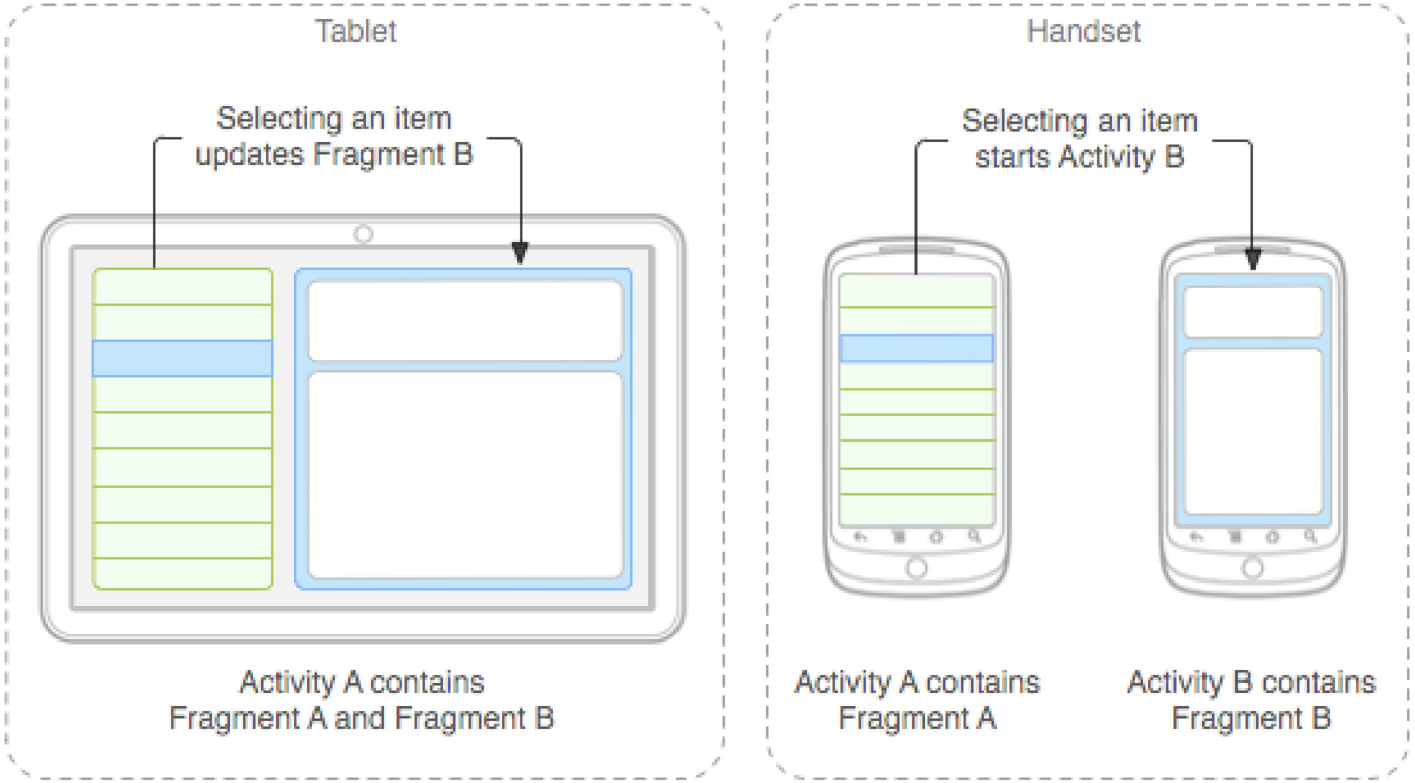
Kian
@naturelvr49
Following
😍 Animal lover, writer, traveller
831 Following 1,807 Followers
Tweets Tweets & replies Media Likes

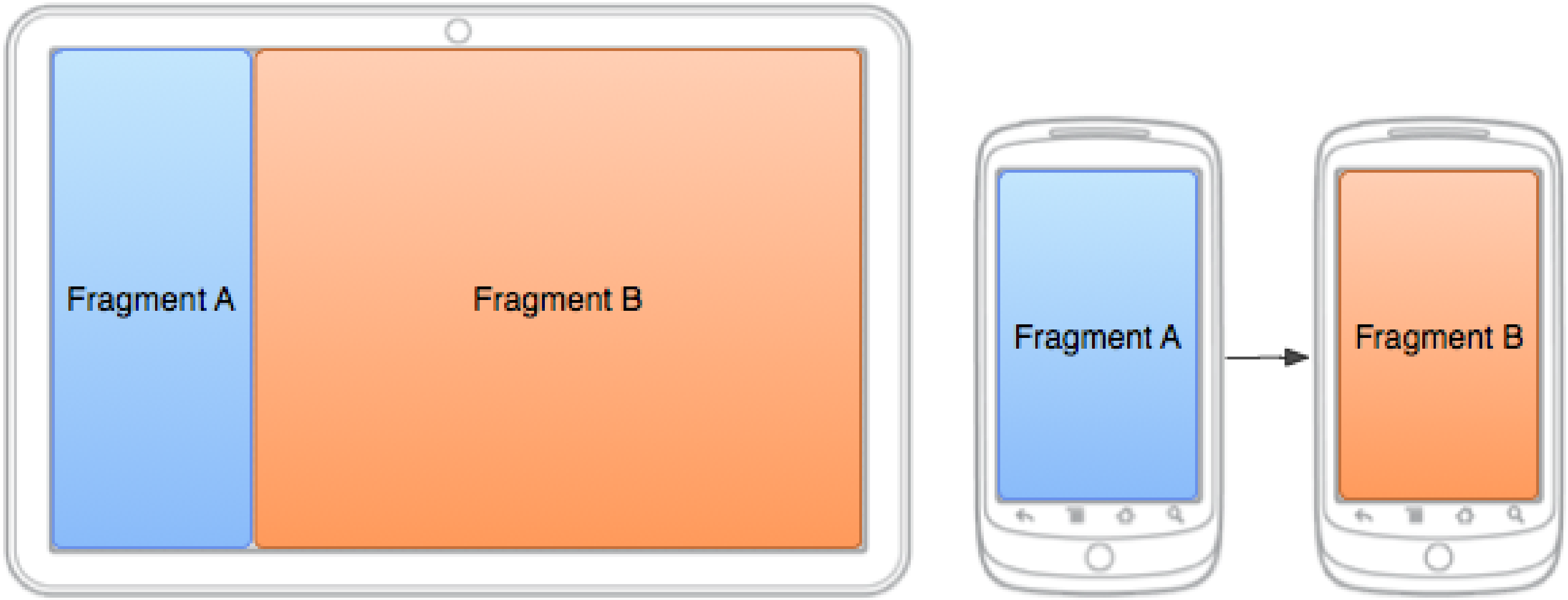
Kian @naturelvr49 · 1m
Feeling a breeze during this #heatwave. Bike ride during sunset 😍😊

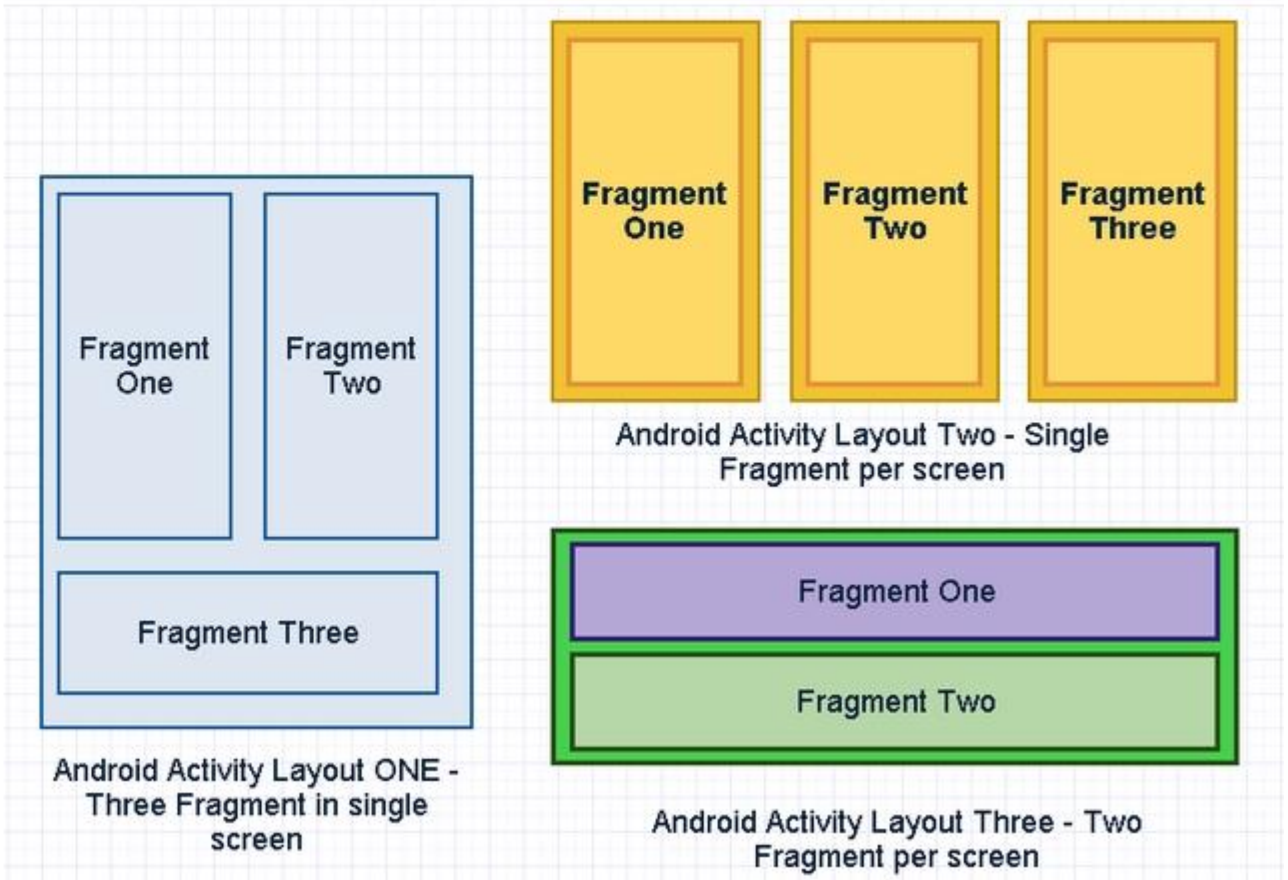
Messaging
Chat privately with friends, followers, businesses, and your government

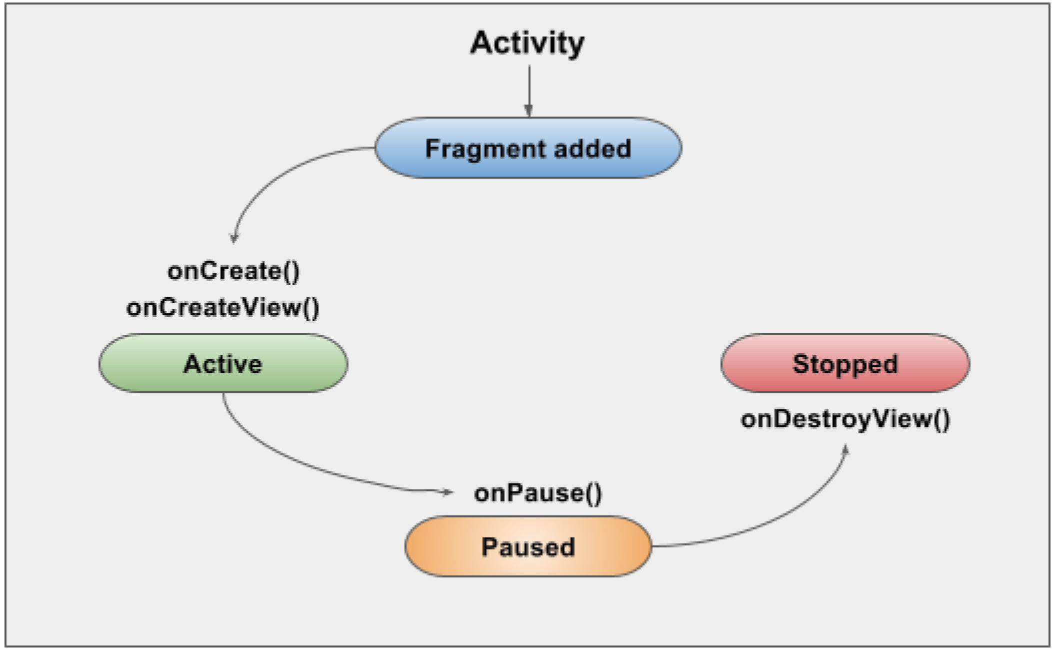
Kian
@naturelvr49

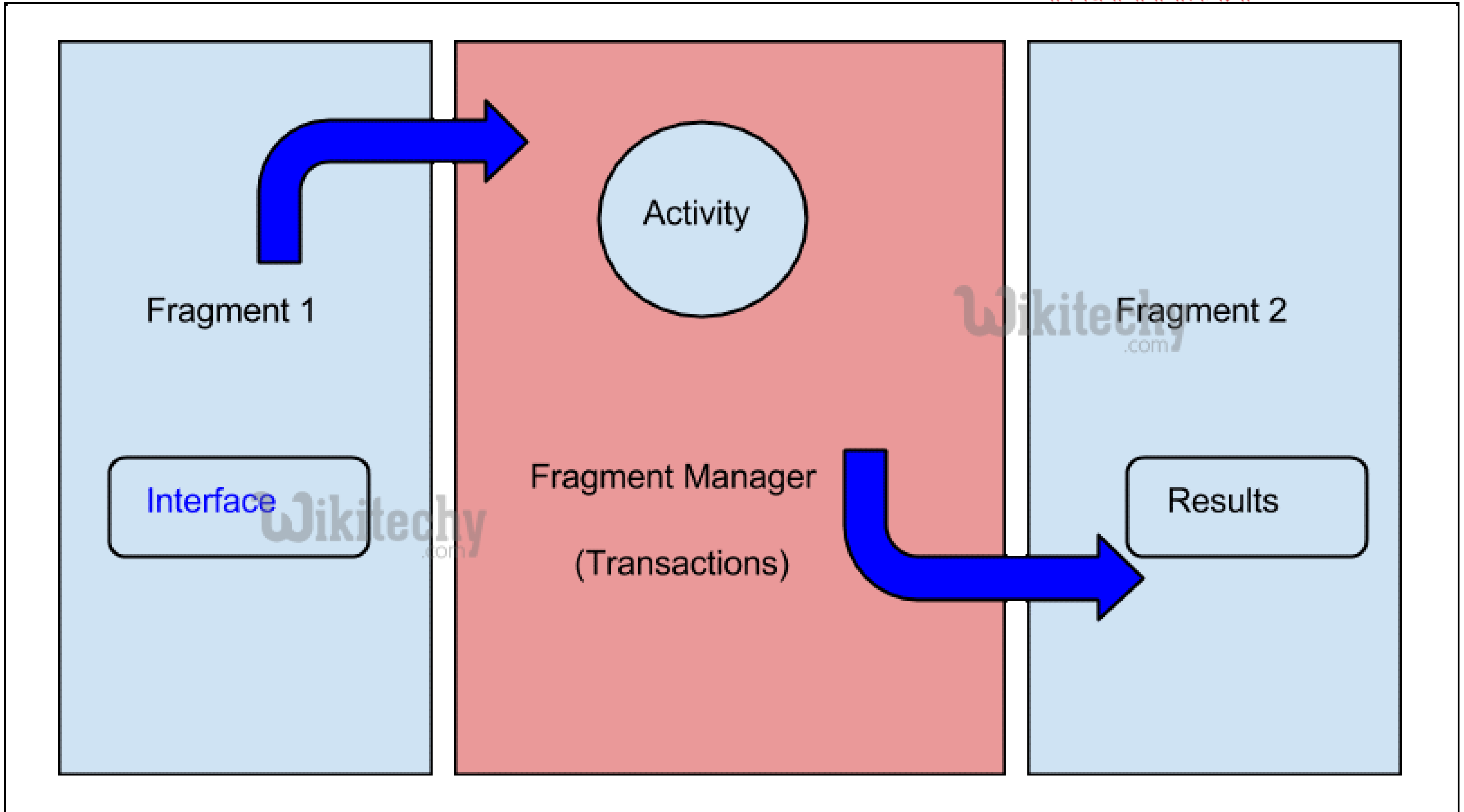
Hi! Plans this weekend?
Kian · 10:57 AM
Going to the beach!
Join me?
10:58 AM ✓
What fun! I'll pack a snack and my sunglasses
Kian · 10:59 AM
Great, can't wait! You're the best :)
11:00 AM ✓
Start a new message

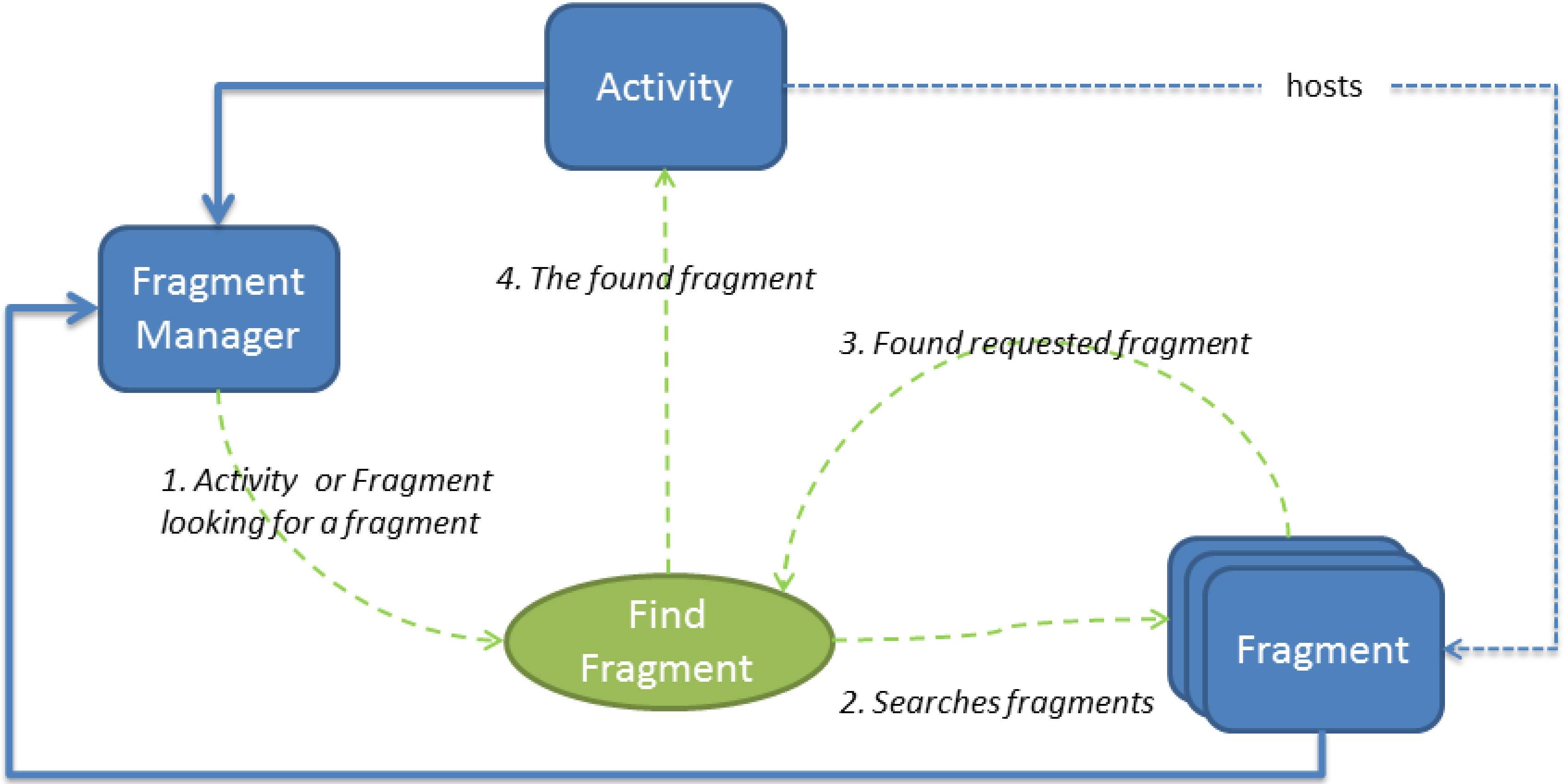

Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

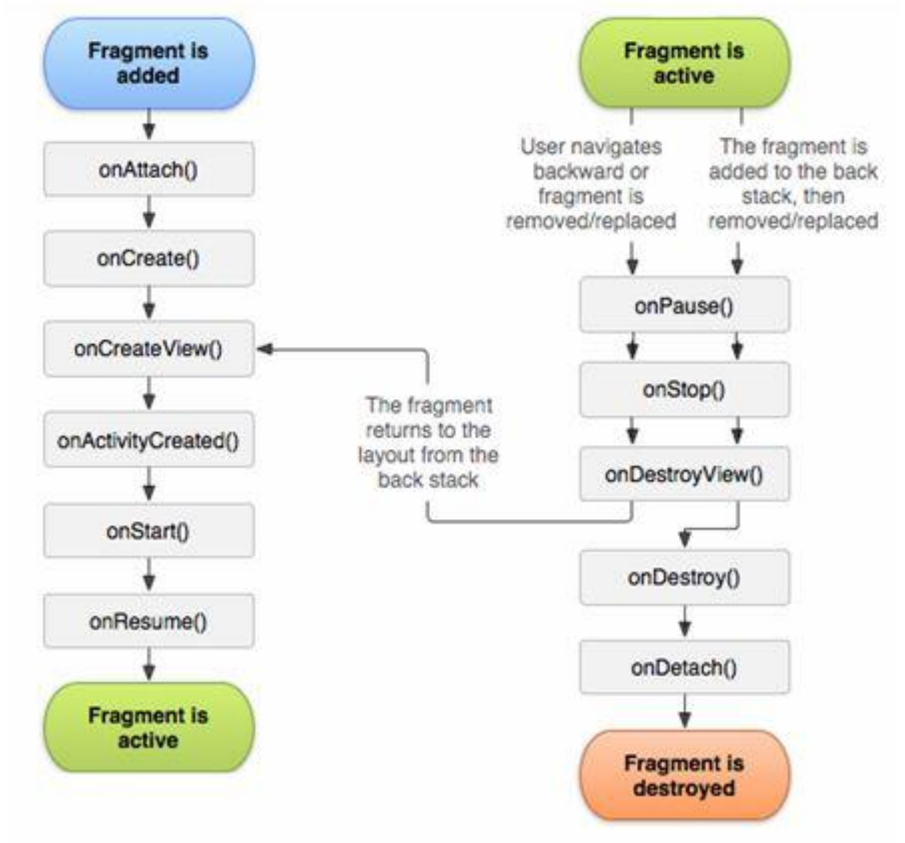

Fragment A
Fragment B
Fragment A
Fragment B


Fragment One
Fragment Two
Fragment Three
Android Activity Layout ONE - Three Fragment in single screen
Fragment One
Fragment Two
Fragment Three
Android Activity Layout Two - Single Fragment per screen
Fragment One
Fragment Two
Android Activity Layout Three - Two Fragment per screen


Activity
Fragment added
onCreate()
onCreateView()
Active
onPause()
Paused
onDestroyView()
Stopped

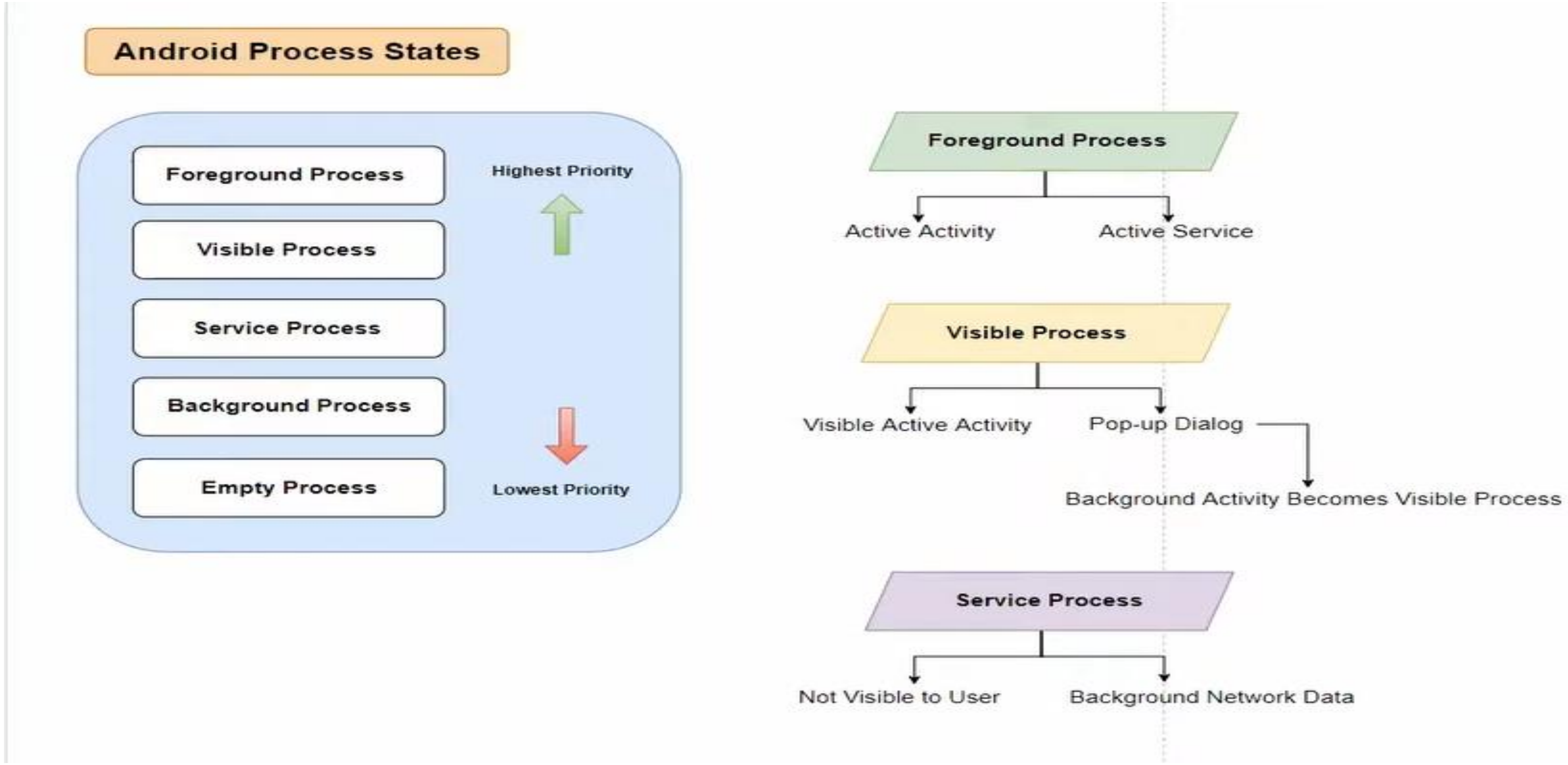

Activity
Fragment 1
Fragment 2
Interface
Fragment Manager
(Transactions)
Results


Activity
hosts
Fragment Manager
4. The found fragment
3. Found requested fragment
1. Activity or Fragment looking for a fragment
Find Fragment
Fragment
2. Searches fragments

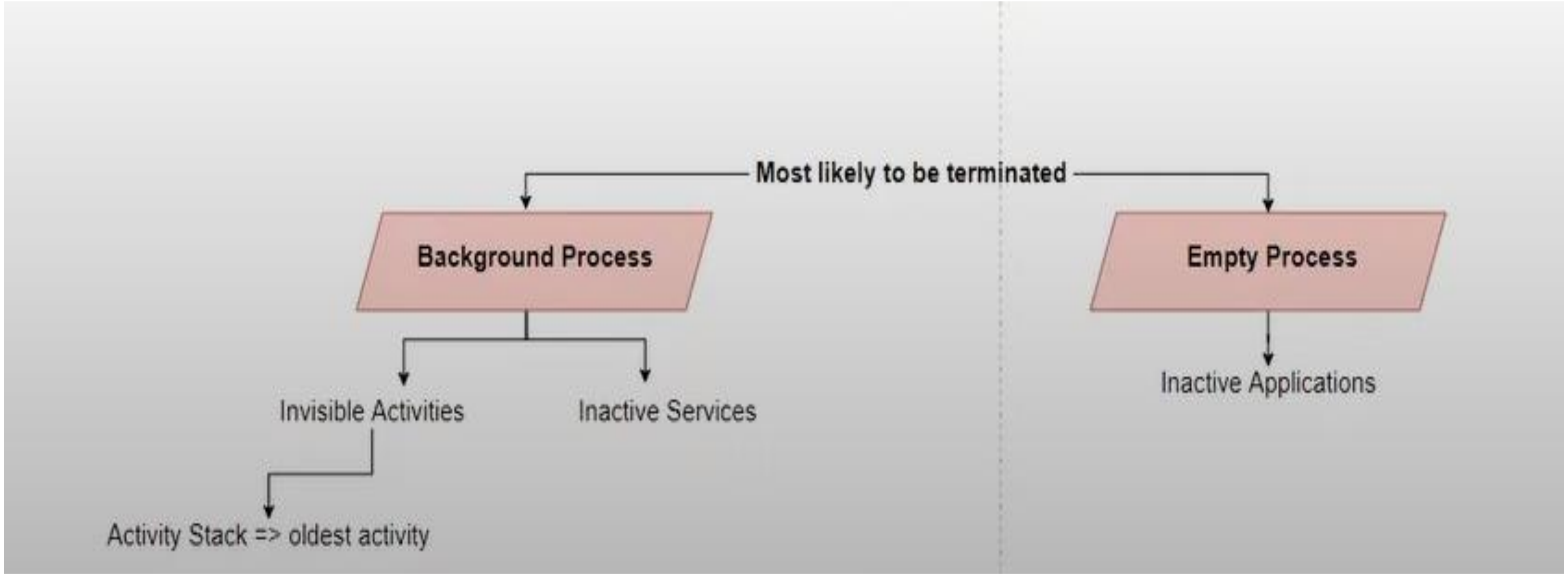


Fragment is added
onAttach()
onCreate()
onCreateView()
onActivityCreated()
onStart()
onResume()
Fragment is active
Fragment is active
User navigates backward or fragment is removed/replaced
The fragment is added to the back stack, then removed/replaced
onPause()
onStop()
onDestroyView()
The fragment returns to the layout from the back stack
onDestroy()
onDetach()
Fragment is destroyed

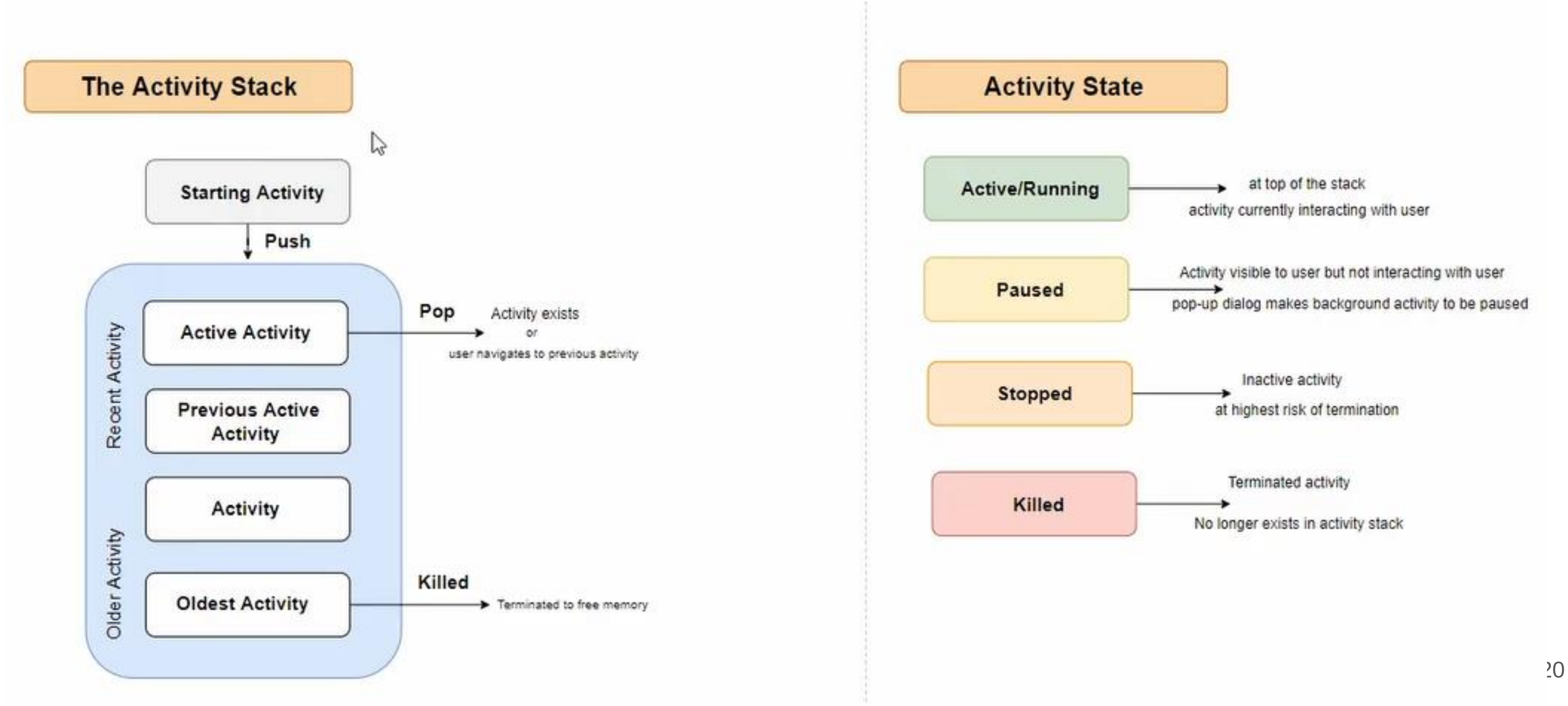
Life Cycle of an Activity:



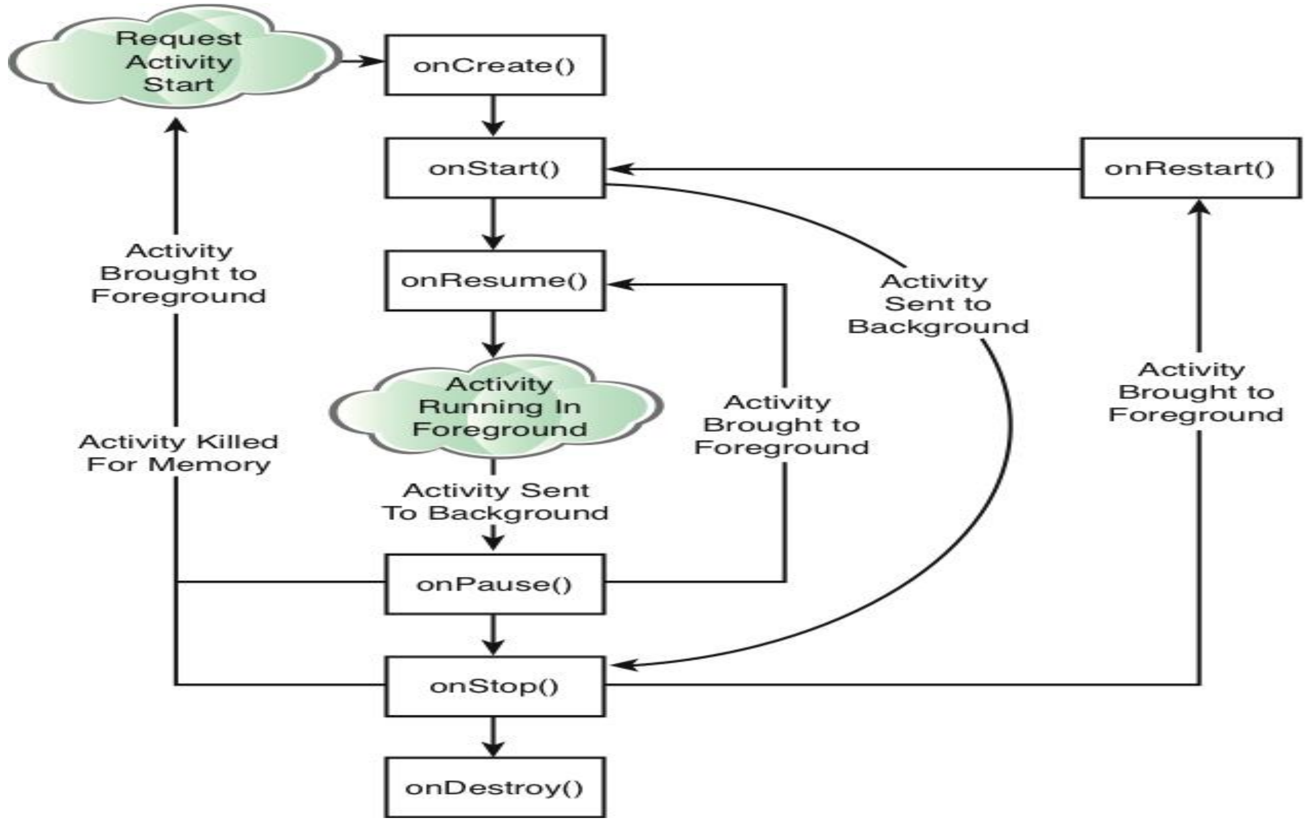
Life Cycle of an Activity:



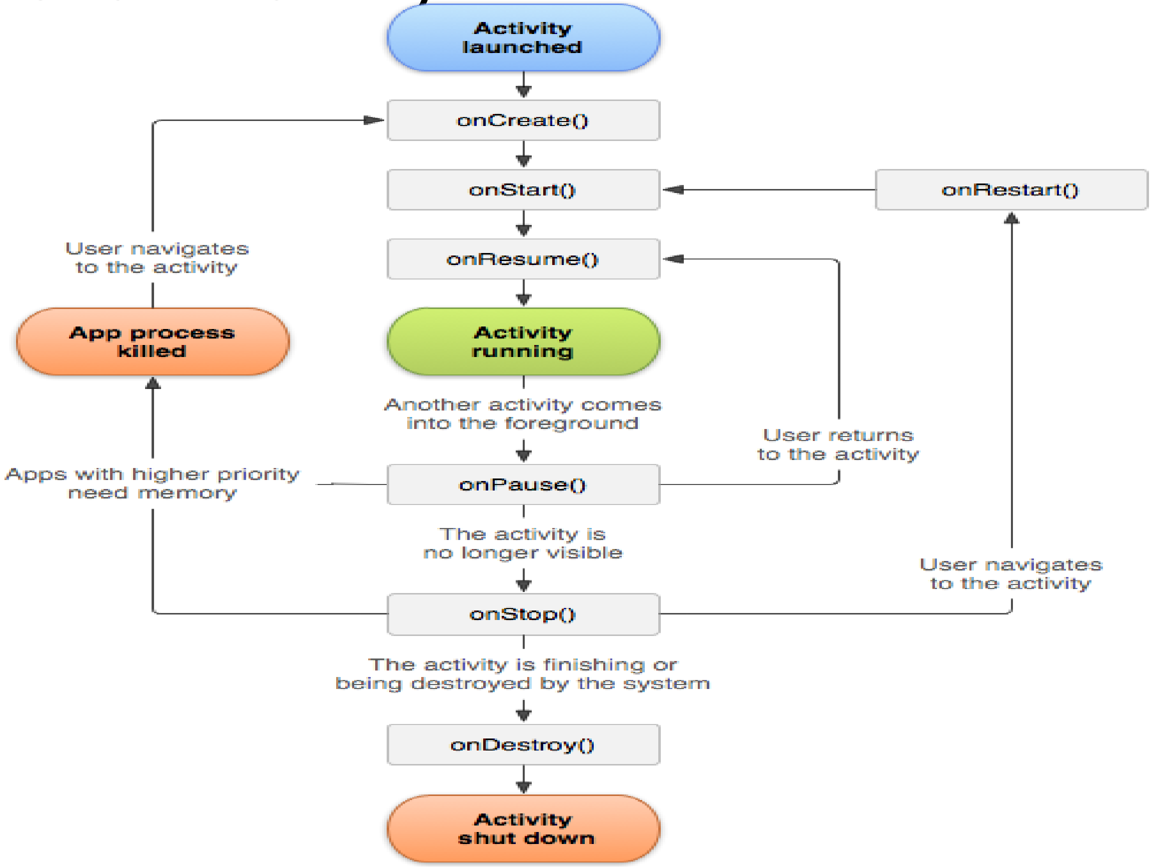
Life Cycle of an Activity:


The Activity Stack
Starting Activity
Push
Active Activity
Previous Active Activity
Activity
Oldest Activity
Recent Activity
Older Activity
Pop
Activity exists
or
user navigates to previous activity
Killed
Terminated to free memory
Activity State
Active/Running
at top of the stack
activity currently interacting with user
Paused
Activity visible to user but not interacting with user
pop-up dialog makes background activity to be paused
Stopped
Inactive activity
at highest risk of termination
Killed
Terminated activity
No longer exists in activity stack

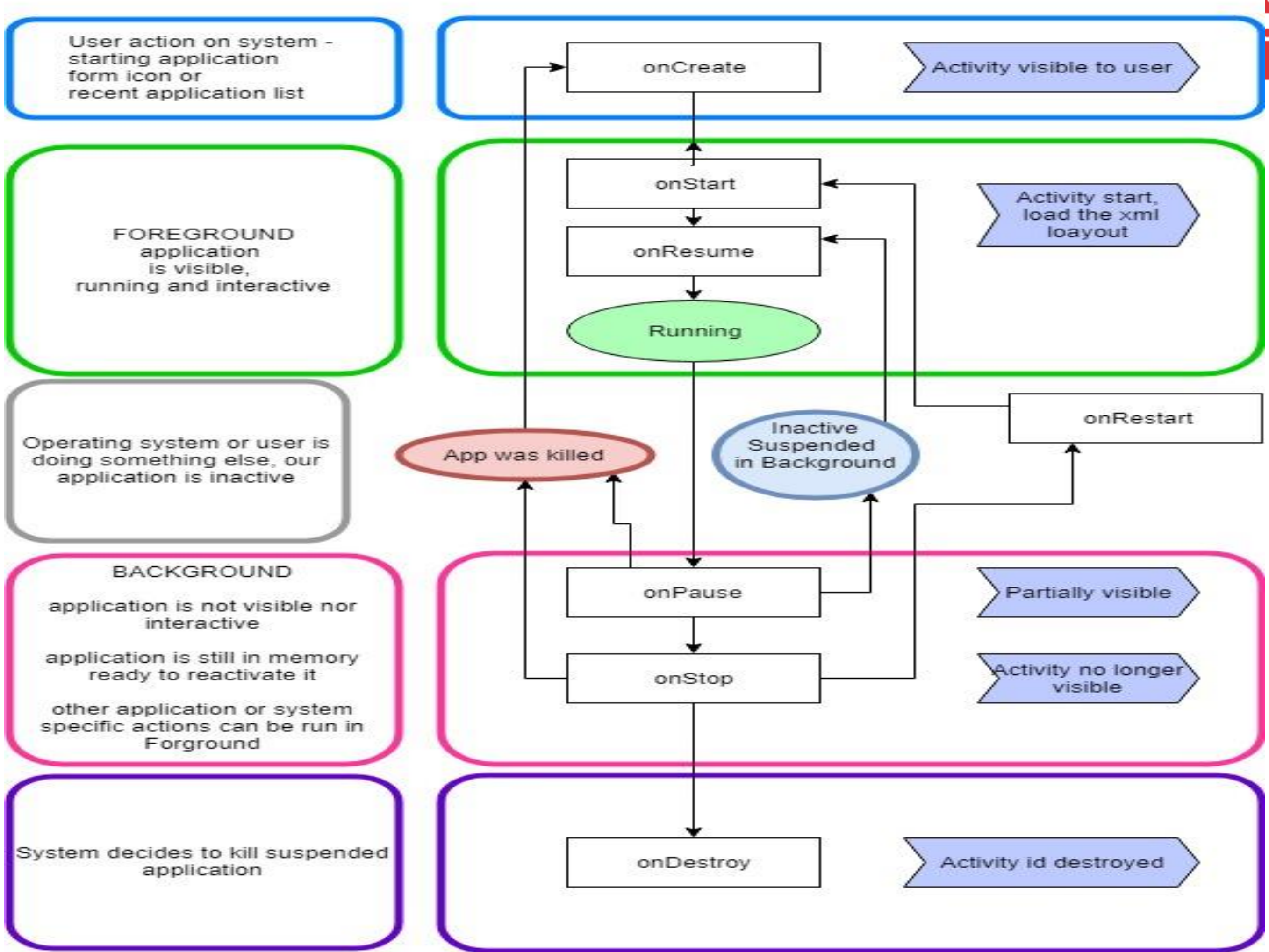
Life Cycle of an Activity:


Request Activity Start
onCreate()
onStart()
onRestart()
onResume()
Activity Brought to Foreground
Activity Sent to Background
Activity Running In Foreground
Activity Brought to Foreground
Activity Brought to Foreground
Activity Killed For Memory
Activity Sent To Background
onPause()
onStop()
onDestroy()

Life Cycle of an Activity:


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

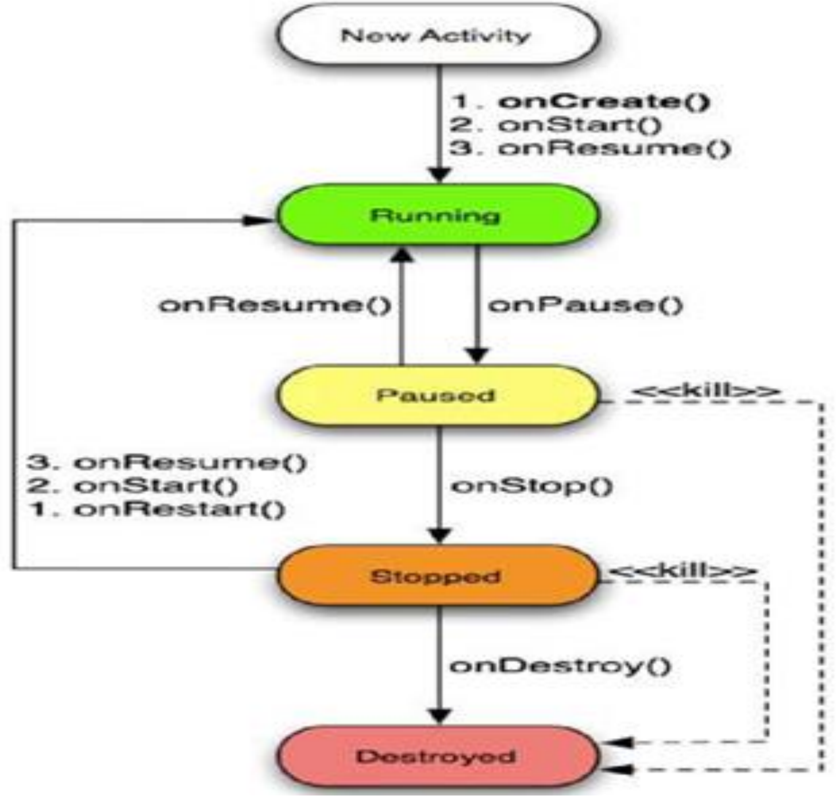
Android


User action on system - starting application form icon or recent application list
onCreate
Activity visible to user
FOREGROUND application is visible, running and interactive
onStart
onResume
Running
Activity start, load the xml loayout
Operating system or user is doing something else, our application is inactive
App was killed
Inactive Suspended in Background
onRestart
BACKGROUND
application is not visible nor interactive
application is still in memory ready to reactivate it
other application or system specific actions can be run in Forground
onPause
Partially visible
onStop
Activity no longer visible
System decides to kill suspended application
onDestroy
Activity id destroyed

Life Cycle States:

It has three states

1. *active / running*
2. *paused or*
3. *stopped*



Life Cycle States

- **Running**
  - It is *active or running when it is in the foreground of the screen(at the top of the activity stack for the current task).*
  - *This is the activity that is the focus for the user's actions.*

Life Cycle States

- **Paused**
  - It is ***paused*** *if it has lost focus but is still visible to the user.*
  - *That is, another activity lies on top of it and that new activity either is transparent or doesn't cover the full screen.*
  - *A paused activity is completely alive(it maintains all state and member information and remains attached to the window manager), but can be killed by the system in extreme low memory situations*

Life Cycle States

- **Stopped**
  - It is ***stopped*** *if it is completely obscured by another activity.*
  - *It still retains all state and member information. However, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere*

Life time

- **Life cycle of an activity:**
- **Visible life time:** onStart() to onStop()
  - During this life time the activity will not be interacting with the user. Between these two activities, resources are maintained that are needed by the system to show the activity on the screen

Life time

- **Foreground life time:** onResume() to onPause().
  - During this life time the activity will be in front of all other activities.
- **Entire Life time:** onCreate() to onDestroy()

Life time

- When you open the app it will go through below states:
- onCreate() –> onStart() –> onResume()
- When you press the back button and exit the app
- onPaused() — > onStop() –> onDestory()

Life time

Ganpat University
॥ विद्यया समाजोत्कर्षः ॥
U.V. Patel College of Engineering

- When you press the home button
- After pressing the home button, again when you open the app from a recent task list
- After dismissing the dialog or back button from the dialog
- ?

Ganpat University ॥ विद्यया समाजोत्कर्षः ॥ | U.V. Patel College of Engineering

Life time

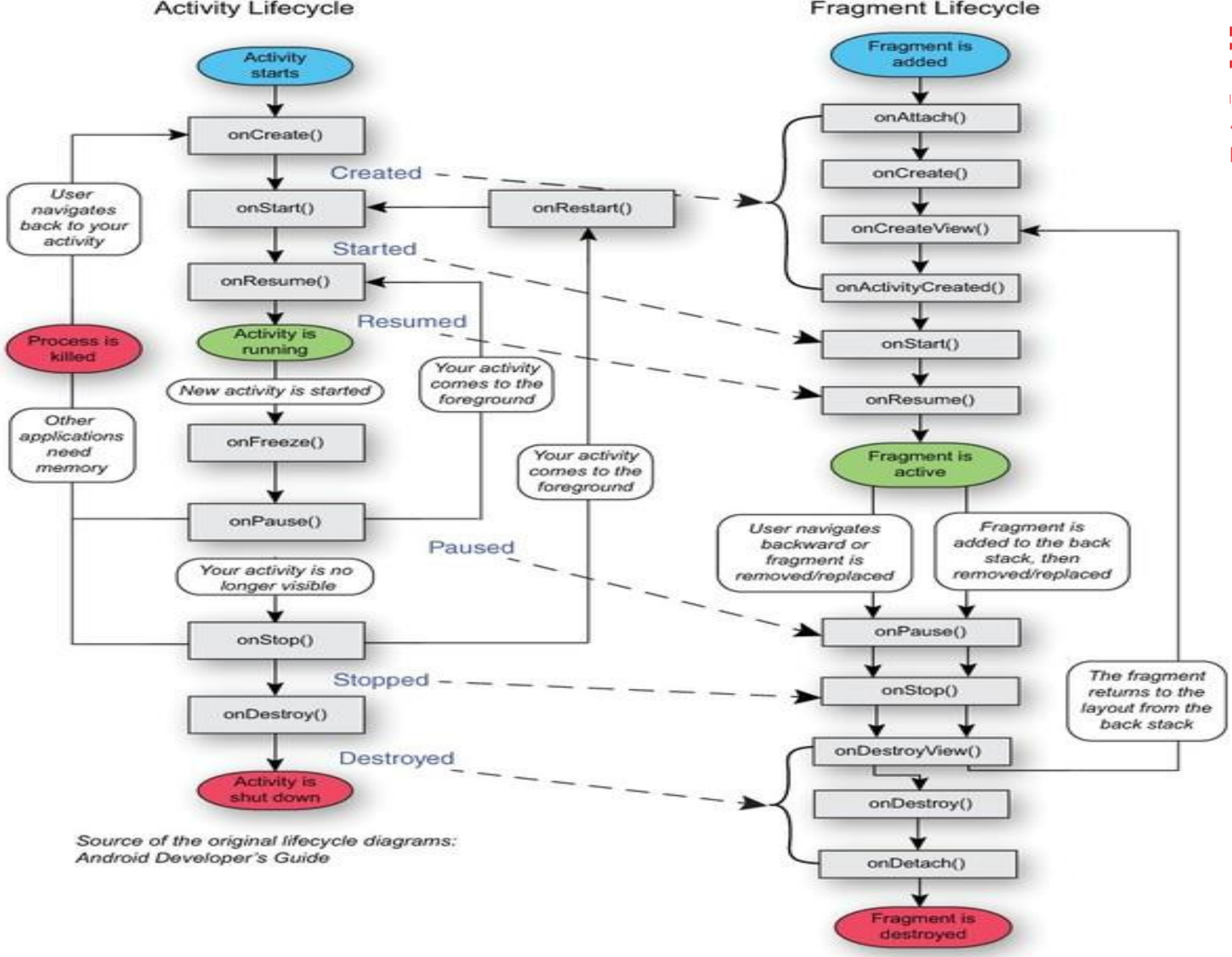
- **When you press the home button**
- onPaused() –> onStop()
- **After pressing the home button, again when you open the app from a recent task list**
- onRestart() –> onStart() –> onResume()
- **After dismissing the dialog or back button from the dialog**
- onResume()

Life time

- If a phone is ringing and user is using the app
- After the call ends
- When your phone screen is off
- When your phone screen is turned back on

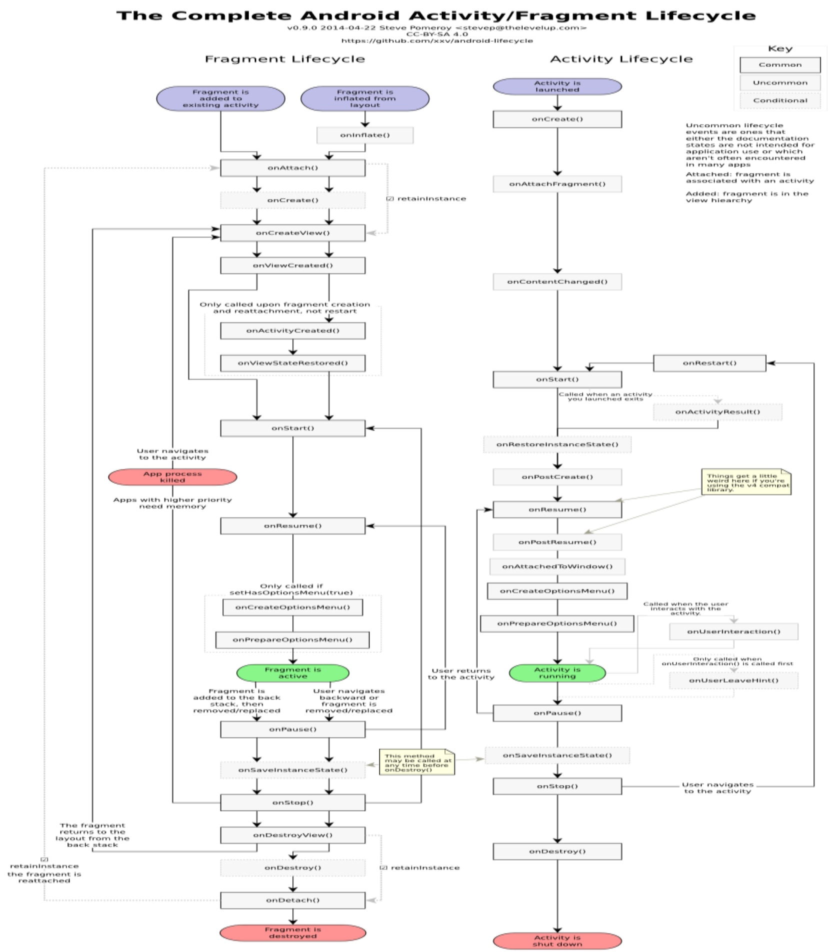
Life time

- **If a phone is ringing and user is using the app**
- onPause() –> onResume()
- **After the call ends**
- onResume()
- **When your phone screen is off**
- onPaused() –> onStop()
- **When your phone screen is turned back on**
- onRestart() –> onStart() –> onResume()



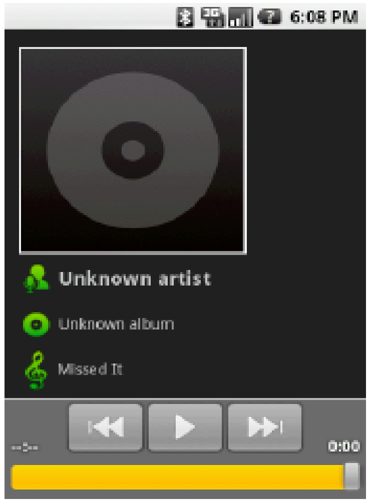
Source of the original lifecycle diagrams: Android Developer's Guide

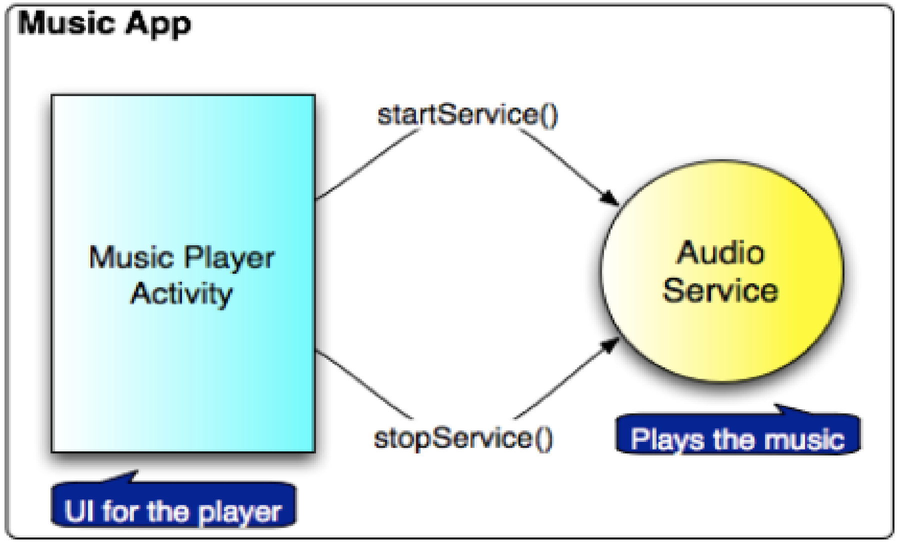
U.V. Patel College of Engineering


The Complete Android Activity/Fragment Lifecycle
v0.9.0 2014-04-22 Steve Pomeroy <stevep@thelevelup.com>
CC-BY-SA 4.0
https://github.com/xxv/android-lifecycle
Fragment Lifecycle
Activity Lifecycle
Key
Common
Uncommon
Conditional
Uncommon lifecycle events are ones that either the documentation states are not intended for application use or which aren't often encountered in many apps
Attached: fragment is associated with an activity
Added: fragment is in the view hiearchy
Fragment is added to existing activity
Fragment is inflated from layout
onInflate()
onAttach()
onCreate()
retainInstance
onCreateView()
onViewCreated()
Only called upon fragment creation and reattachment, not restart
onActivityCreated()
onViewStateRestored()
onStart()
User navigates to the activity
App process killed
Apps with higher priority need memory
onResume()
Only called if setHasOptionsMenu(true)
onCreateOptionsMenu()
onPrepareOptionsMenu()
Fragment is active
Fragment is added to the back stack, then removed/replaced
User navigates backward or fragment is removed/replaced
User returns to the activity
onPause()
onSaveInstanceState()
This method may be called at any time before onDestroy()
onStop()
The fragment returns to the layout from the back stack
onDestroyView()
retainInstance the fragment is reattached
onDestroy()
retainInstance
onDetach()
Fragment is destroyed
Activity is launched
onCreate()
onAttachFragment()
onContentChanged()
onRestart()
onStart()
Called when an activity you launched exits
onActivityResult()
onRestoreInstanceState()
onPostCreate()
Things get a little weird here if you're using the v4 compat library.
onResume()
onPostResume()
onAttachedToWindow()
onCreateOptionsMenu()
onPrepareOptionsMenu()
Called when the user interacts with the activity.
onUserInteraction()
Activity is running
Only called when onUserInteraction() is called first
onUserLeaveHint()
onPause()
onSaveInstanceState()
onStop()
User navigates to the activity
onDestroy()
Activity is shut down

Services

Services are code that runs in the background. They can be started and stopped. Services doesn't have UI.



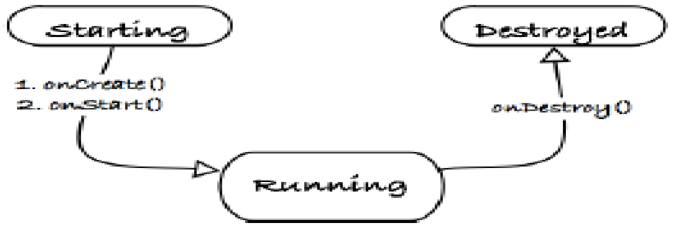


Service Life Cycle

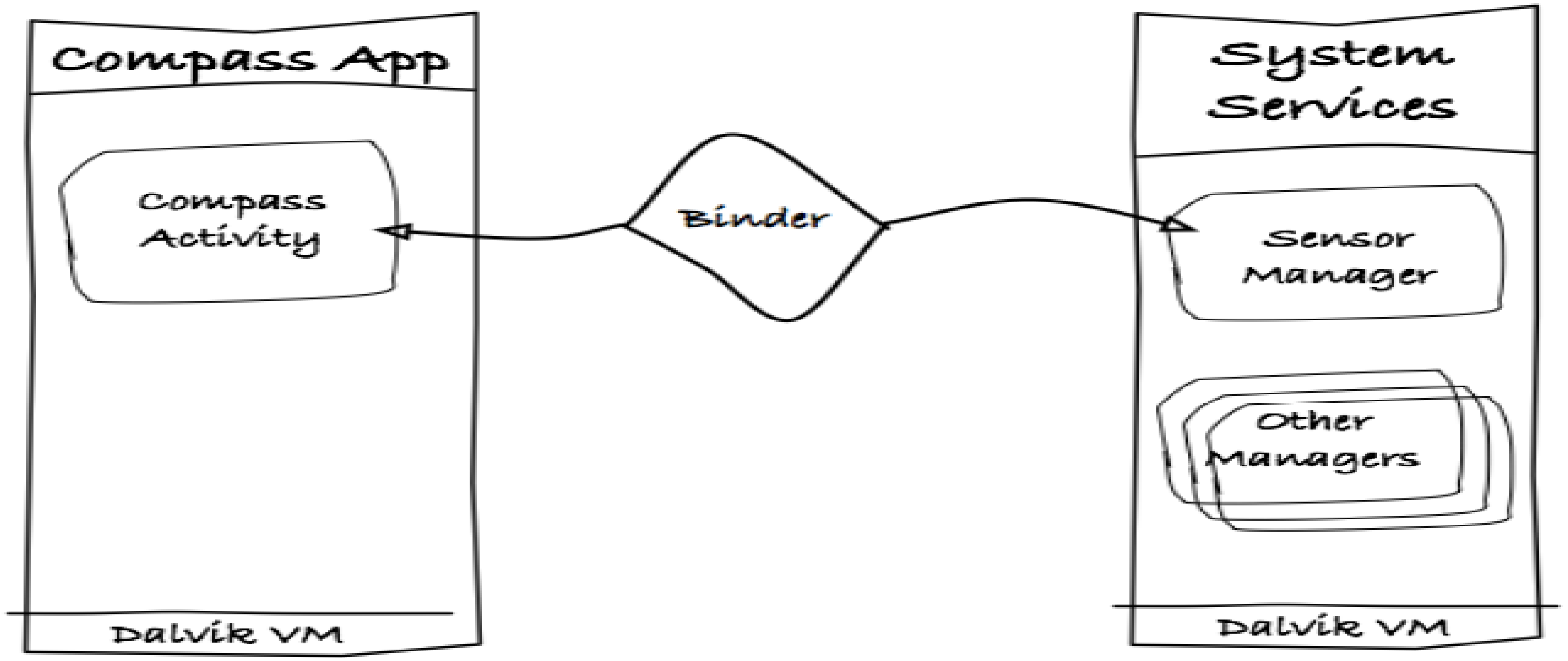
Service also has a lifecycle, but it's much simpler than activity's.

An activity typically starts and stops a service to do some work for it in the background, such as play music, check for new tweets, etc.

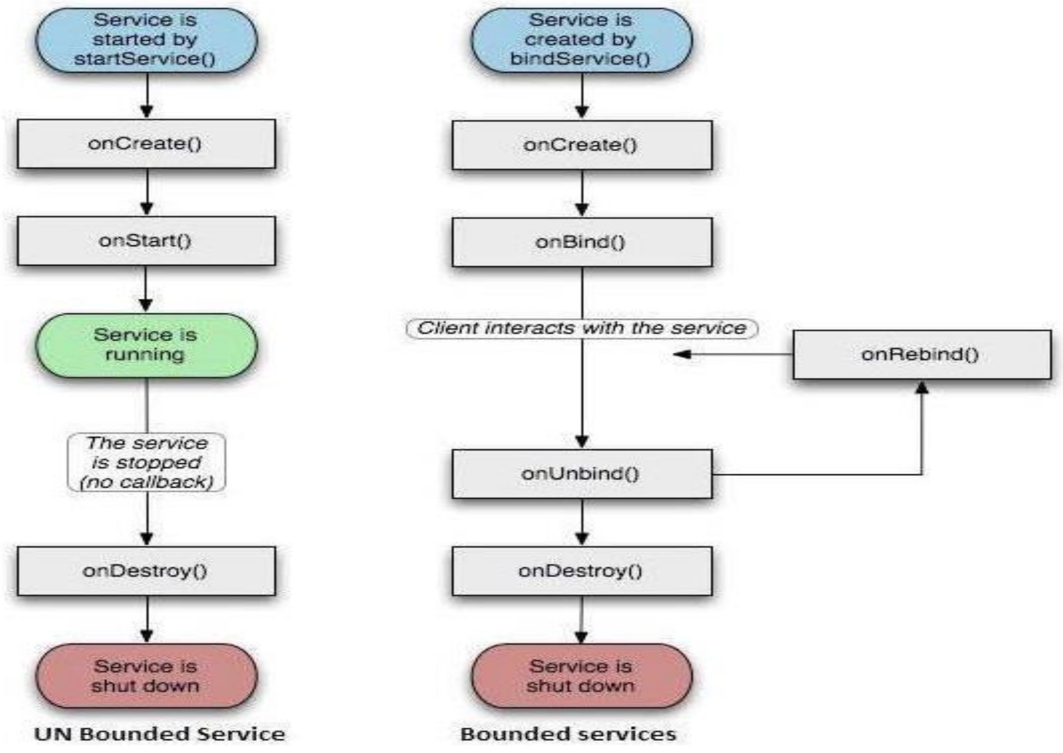
Services can be bound or unbound.



Remote Service



Service Life cycle



Service methods

- **onStartCommand()**

The system calls this method when another component, such as an activity, requests that the service be started, by calling ***startService()*****.** If you implement this method, it is your responsibility to stop the service when its work is done, by calling ***stopSelf()*** or ***stopService()*** methods.

Service methods

- **onBind()**

The system calls this method when another component wants to bind with the service by calling *bindService()*. If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an *IBinder* object. You must always implement this method, but if you don't want to allow binding, then you should return *null*.

Service Life cycle

- **onUnbind()**

The system calls this method when all clients have disconnected from a particular interface published by the service.

- **onRebind()**

The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its *onUnbind(Intent)*.

Service Life cycle

- **onCreate()**

The system calls this method when the service is first created using *onStartCommand()* or *onBind()*. This call is required to perform one-time set-up.

- **onDestroy()**

The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.

Read more about Service

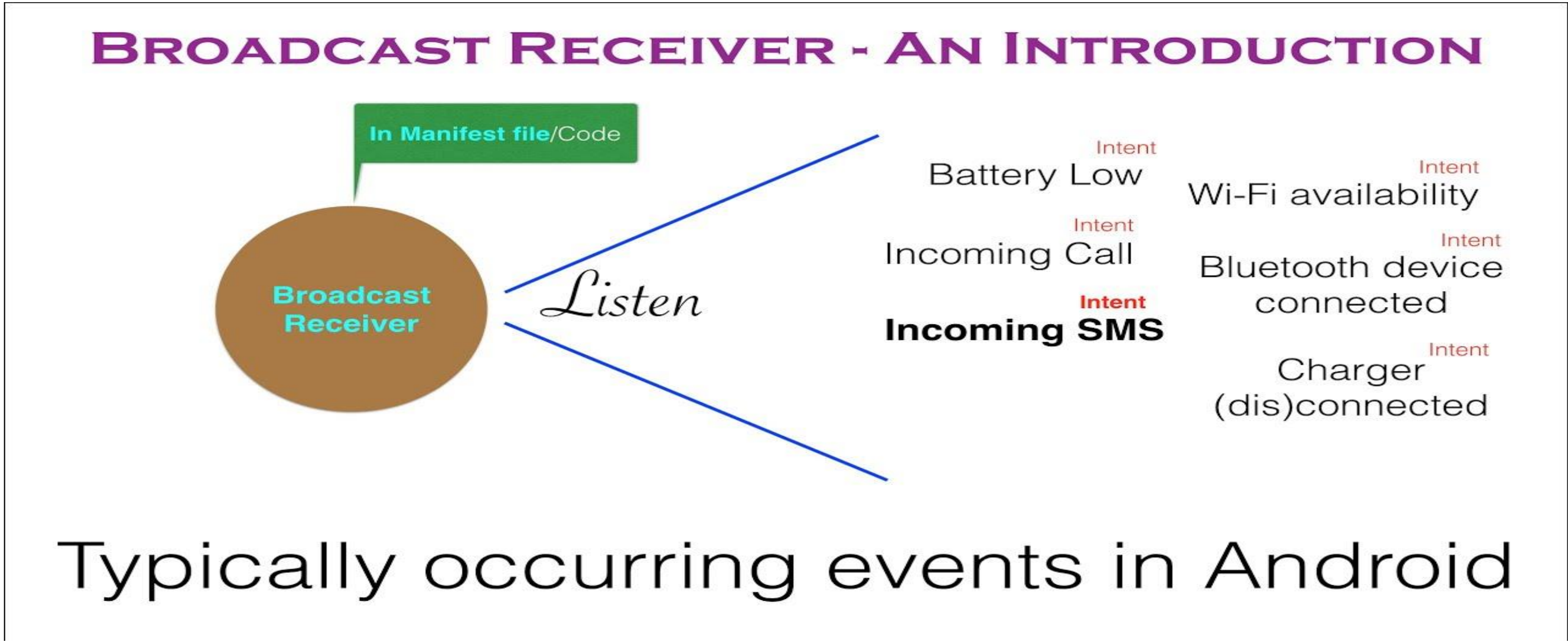
Broadcast Receiver

- Android BroadcastReceiver is a dormant component of android that listens to system-wide broadcast events or intents.

- When any of these events occur it brings the application into action by either creating a status bar notification or performing a task.

- Unlike activities, android BroadcastReceiver doesn’t contain any user interface. Broadcast receiver is generally implemented to delegate the tasks to services depending on the type of intent data that’s received.

Broadcast Receiver



Broadcast Receiver - An Introduction

Typically occurring events in Android

Broadcast Receiver

- Following are some of the important system wide generated intents.

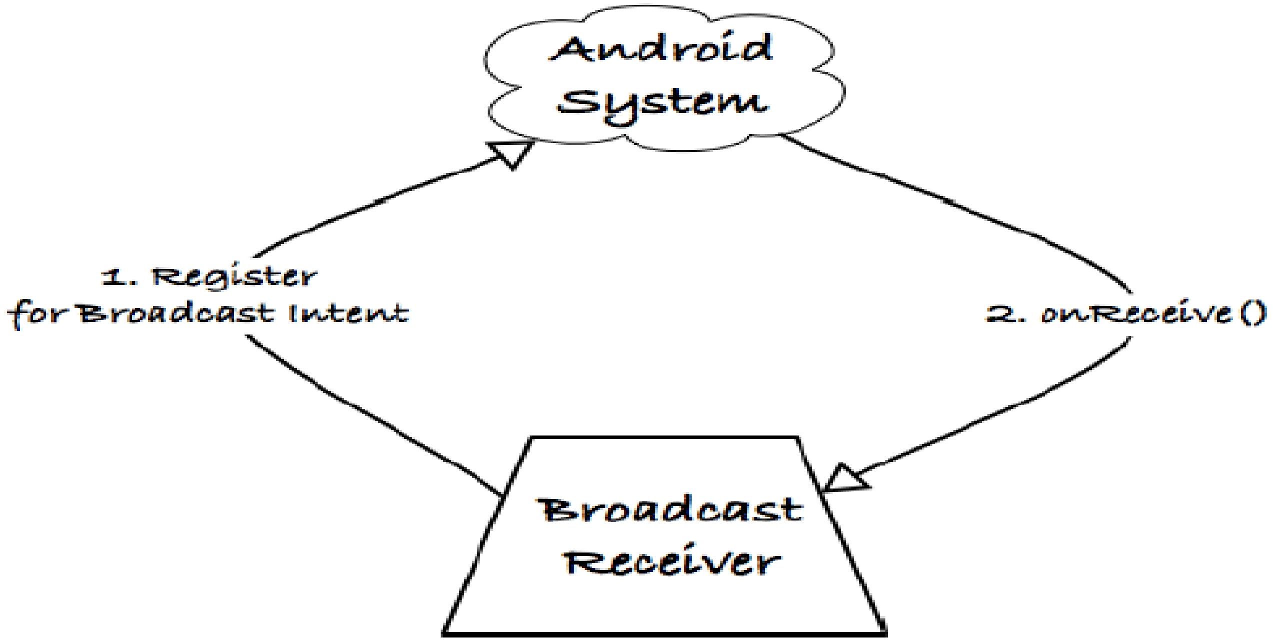
1. **android.intent.action.BATTERY_LOW** : Indicates low battery condition on the device.
2. **android.intent.action.BOOT_COMPLETED** : This is broadcast once, after the system has finished booting
3. **android.intent.action.CALL** : To perform a call to someone specified by the data
4. **android.intent.action.DATE_CHANGED** : The date has changed
5. **android.intent.action.REBOOT** : Have the device reboot
6. **android.net.conn.CONNECTIVITY_CHANGE** : The mobile network or wifi connection is changed(or reset)

Broadcast Receiver in Android

- To set up a Broadcast Receiver in android application we need to do the following two things:

1. Creating a BroadcastReceiver
2. Registering a BroadcastReceiver

Broadcast Receiver



An Intent-based publish-subscribe mechanism. Great for listening system events such as SMS messages.

Broadcast Receiver

Creating the Broadcast Receiver

A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and overriding the onReceive() method where each message is received as a **Intent** object parameter.

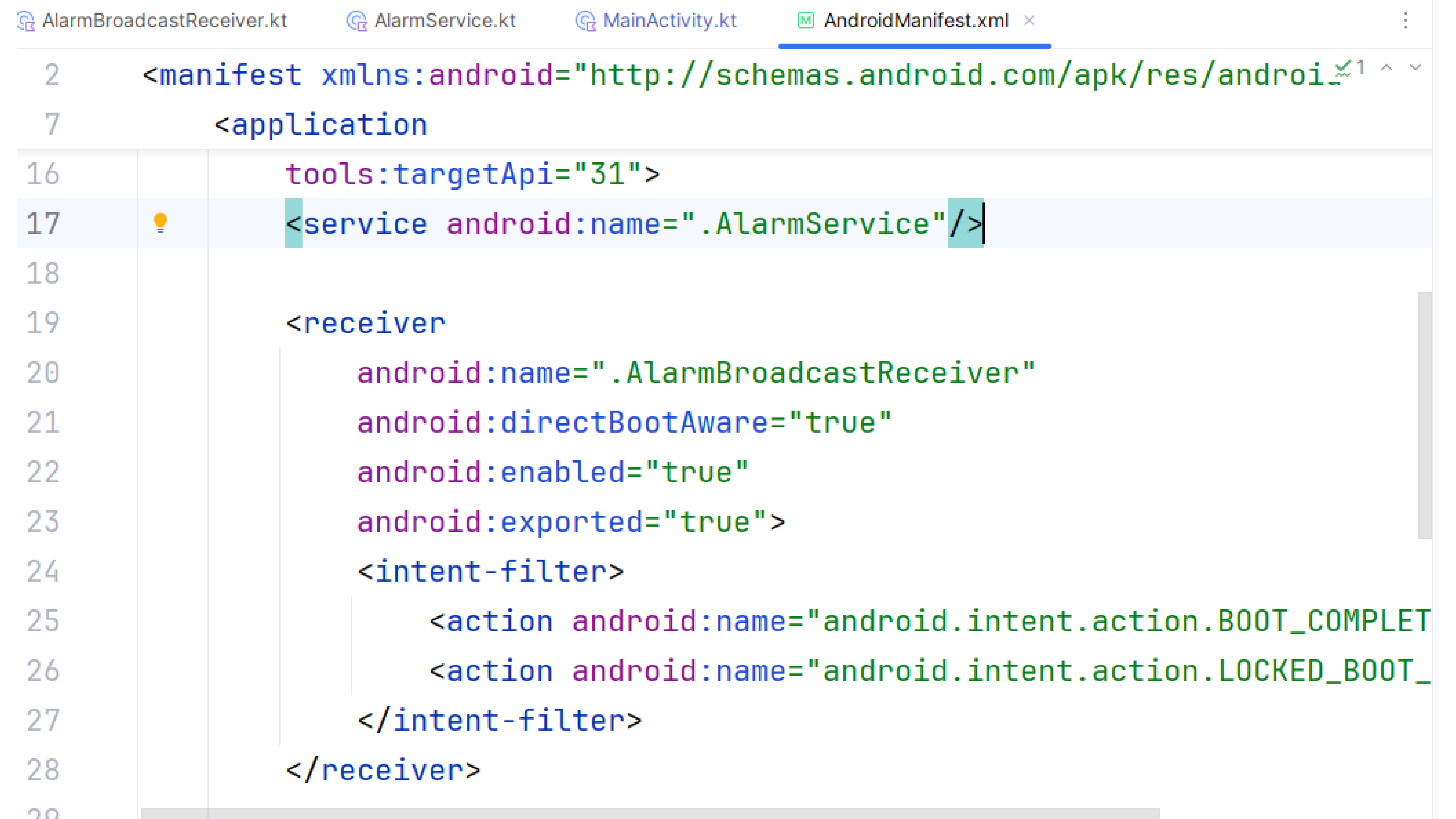
```
import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent

class MyReceiver : BroadcastReceiver() {

    override fun onReceive(context: Context, intent: Intent) {
        // TODO: This method is called when the BroadcastReceiver is receiving
        // an Intent broadcast.
        throw UnsupportedOperationException("Not yet implemented")
    }
}
```

Broadcast Receiver

Registering Broadcast Receiver

An application listens for specific broadcast intents by registering a broadcast receiver in *AndroidManifest.xml* file. Consider we are going to register *MyReceiver* for system generated event ACTION_BOOT_COMPLETED which is fired by the system once the Android system has completed the boot process.

```
<application android:icon="@drawable/ic_launcher" android:label="@string/app_name"
android:theme="@style/AppTheme" >
    <receiver android:name="MyReceiver">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED">
        </action>
        </intent-filter>
    </receiver>
</application>
```



```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/androi
    <application
        tools:targetApi="31">
        <service android:name=".AlarmService"/>

        <receiver
            android:name=".AlarmBroadcastReceiver"
            android:directBootAware="true"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLET
                <action android:name="android.intent.action.LOCKED_BOOT_
            </intent-filter>
        </receiver>
```

TextClock

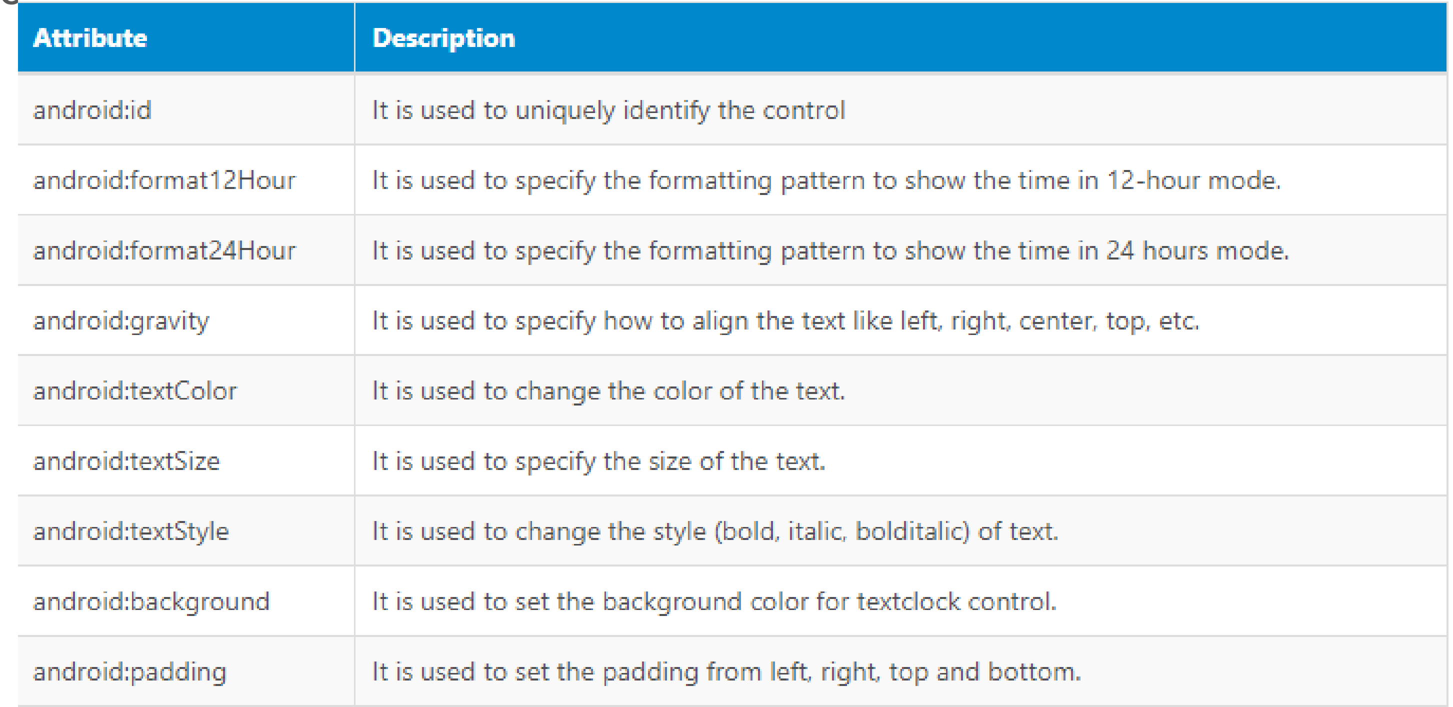
- In android, TextClock is a UI control that is used to show the current date or time as a formatted string.
- The TextClock provides a date/time in two formats, one is to show the date/time in 24 Hours format and another one is to show the date / time in 12-hour format.

- By using the is24HourModeEnabled() method, we can easily know whether the system using TextClock in 24 Hours format or 12 Hours format.

- Following is the pictorial representation of using a TextClock in android applications.
- The android TextClock has been introduced in API Level 17 so if we use

Android TextClock Control Attributes

- The following are some of the commonly used attributes related to TextClock control in android applications



Attribute	Description
android:id	It is used to uniquely identify the control
android:format12Hour	It is used to specify the formatting pattern to show the time in 12-hour mode.
android:format24Hour	It is used to specify the formatting pattern to show the time in 24 hours mode.
android:gravity	It is used to specify how to align the text like left, right, center, top, etc.
android:textColor	It is used to change the color of the text.
android:textSize	It is used to specify the size of the text.
android:textStyle	It is used to change the style (bold, italic, bolditalic) of text.
android:background	It is used to set the background color for textclock control.
android:padding	It is used to set the padding from left, right, top and bottom.

Real Time Clock

```
<TextClock
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:format12Hour="hh:mm:ss a MMM,dd yyyy"/>
```

Calendar

- The Calendar class is an abstract class that provides methods for converting between a specific instant in time and a set of calendar fields such as YEAR, MONTH, DAY_OF_MONTH, HOUR, and so on, and for manipulating the calendar fields, such as getting the date of the next week. An instant in time can be represented by a millisecond value that is an offset from the Epoch, January 1, 1970 00:00:00.000 GMT (Gregorian).

- The class also provides additional fields and methods for implementing a concrete calendar system outside the package. Those fields and methods are defined as protected.

- Like other locale-sensitive classes, Calendar provides a class method, getInstance, for getting a generally useful object of this type. Calendar's

Date/Time

```
fun getCurrentDateTime(): String {
    val cal = Calendar.getInstance()
    val df: DateFormat = SimpleDateFormat("MMM, dd yyyy hh:mm:ss a")
    return df.format(cal.time)
}
fun getMillis(hour:Int,min:Int):Long
{
    val setcalendar = Calendar.getInstance()
    setcalendar[Calendar.HOUR_OF_DAY] = hour
    setcalendar[Calendar.MINUTE] = min
    setcalendar[Calendar.SECOND] = 0
    return setcalendar.timeInMillis
}
```

Date/Time

```
fun getHour():Int{
    val cal = Calendar.getInstance()
    cal.time = Date(remindertime)
    return cal[Calendar.HOUR_OF_DAY]
}
fun getMinute():Int{
    val cal = Calendar.getInstance()
    cal.time = Date(remindertime)
    return cal[Calendar.MINUTE]
}
```

Ganpat University | U.V. Patel College of Engineering
॥ विद्यया समाजोत्कर्षः ॥

Switch & Time picker

Switch Material Design

```
<com.google.android.material.switchmaterial.SwitchMaterial
    android:id="@+id/reminderSwitch"
    android:layout_width="0dp"
    android:gravity="end|center_vertical"
    android:textSize="30sp"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Set Reminder" />
```

Time Picker

```xml
<TimePicker
    android:id="@+id/reminderTime"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/layoutReminder" />
```

Property of Switch & Time Picker

```
val timePicker = customLayout.findViewById<TimePicker>(R.id.reminderTime)
reminderSwitch.isChecked = note.isReminder
timePicker.hour = note.getHour()
timePicker.minute = note.getMinute()
```

Show Time Picker Dialog

```
fun showTimerDialog()
{
    val cldr: Calendar = Calendar.getInstance()
    val hour: Int = cldr.get(Calendar.HOUR_OF_DAY)
    val minutes: Int = cldr.get(Calendar.MINUTE)
    // time picker dialog
    val picker = TimePickerDialog(
        this@DashboardActivity,
        { tp, sHour, sMinute -> sendDialogDataToActivity(sHour, sMinute) },
        hour,
        minutes,
        false
    )

    picker.show()
}
```

Send to set alarm

```
private fun sendDialogDataToActivity(hour: Int, minute: Int) {
    val alarmCalendar = Calendar.getInstance()
    val year: Int = alarmCalendar.get(Calendar.YEAR)
    val month: Int = alarmCalendar.get(Calendar.MONTH)
    val day: Int = alarmCalendar.get(Calendar.DATE)
    alarmCalendar.set(year, month, day, hour, minute, 0)
    textAlarmTime.text = SimpleDateFormat("hh:mm ss a").format(alarmCalendar.time)
    setAlarm(alarmCalendar.timeInMillis, "Start")
    Toast.makeText(
        this,
        "Time: hours:${hour}, minutes:${minute},
millis:${alarmCalendar.timeInMillis}",
        Toast.LENGTH_SHORT
    ).show()
}
```

PendingIntent

- By giving a PendingIntent to another application, you are granting it the right to perform the operation you have specified as if the other application was yourself (with the same permissions and identity).
- Instances of this class are created with getActivity(Context, int, Intent, int), getActivities(Context, int, Intent[], int), getBroadcast(Context, int, Intent, int), and getService(Context, int, Intent, int);
- The returned object can be handed to other applications so that they can perform the action you described on your behalf at a later time.
- public static PendingIntent getBroadcast (Context context,

```
                int requestCode,
                Intent intent,
                int flags)
```

AlarmManager

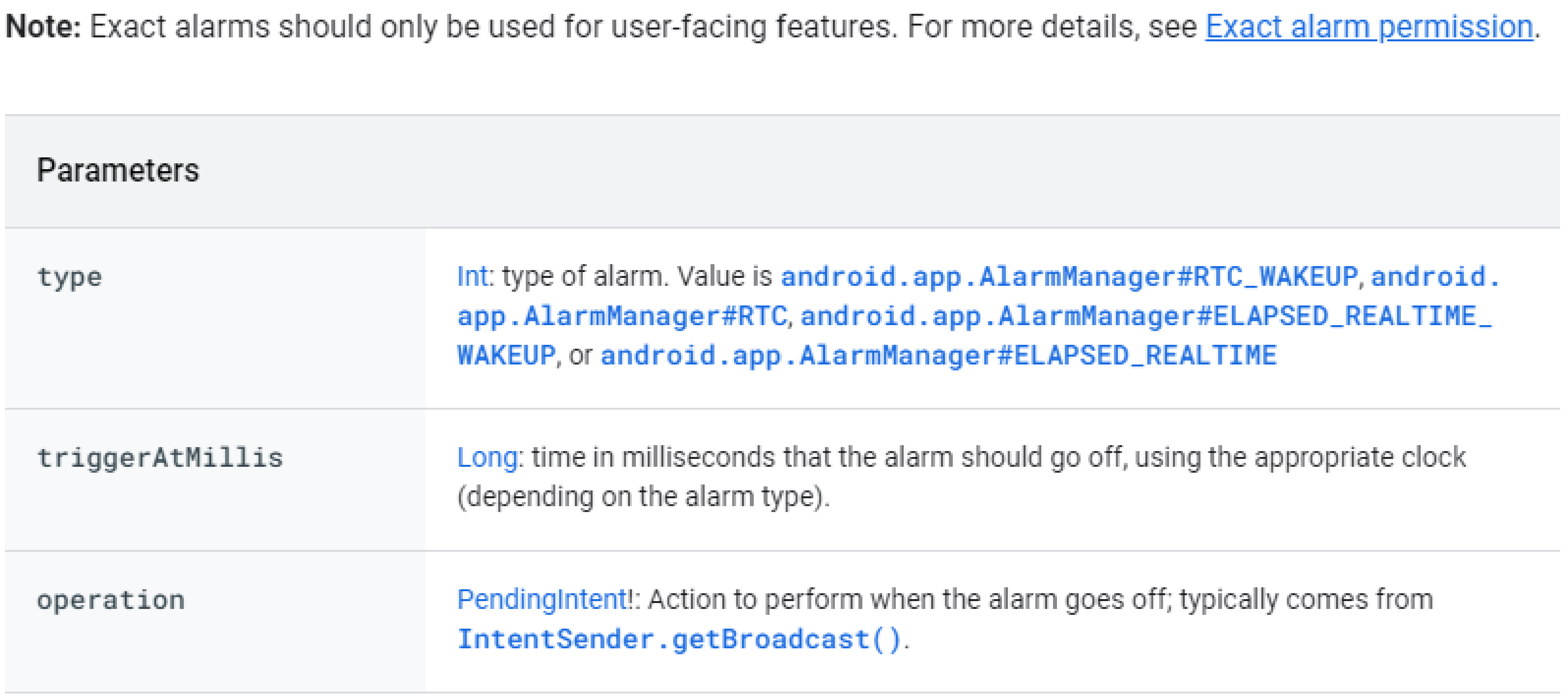
- This class provides access to the system alarm services.
- These allow you to schedule your application to be run at some point in the future.
- When an alarm goes off, the Intent that had been registered for it is broadcast by the system, automatically starting the target application if it is not already running.
- Registered alarms are retained while the device is asleep (and can optionally wake the device up if they go off during that time), but will be cleared if it is turned off and rebooted.
- The Alarm Manager is intended for cases where you want to have your application code run at a specific time, even if your application is not currently running. For normal timing operations (ticks, timeouts, etc) it is easier.

Ganpat University U.V. Patel College of Engineering
॥ विद्यया समाजोत्कर्षः ॥

AlarmManager - setExact()

- open fun setExact(
  type: Int,
  triggerAtMillis: Long,
  operation: PendingIntent!
): Unit
- Schedule an alarm to be delivered precisely at the stated time.
- This method is like set(int,long,android.app.PendingIntent), but does not permit the OS to adjust the delivery time. The alarm will be delivered as nearly as possible to the requested trigger time.
- Only alarms for which there is a strong demand for exact-time delivery (such as an alarm clock ringing at the requested time) should be scheduled as exact. Applications are strongly discouraged from using exact alarms unnecessarily as they reduce the OS's ability to minimize battery use.

AlarmManager - setExact()



Note: Exact alarms should only be used for user-facing features. For more details, see Exact alarm permission.

Parameters	
`type`	Int: type of alarm. Value is `android.app.AlarmManager#RTC_WAKEUP`, `android.app.AlarmManager#RTC`, `android.app.AlarmManager#ELAPSED_REALTIME_WAKEUP`, or `android.app.AlarmManager#ELAPSED_REALTIME`
`triggerAtMillis`	Long: time in milliseconds that the alarm should go off, using the appropriate clock (depending on the alarm type).
`operation`	PendingIntent!: Action to perform when the alarm goes off; typically comes from `IntentSender.getBroadcast()`.

Set Alarm

```kotlin
fun setAlarm(millisTime: Long, str: String)
{
    val intent = Intent(this, AlarmBroadcastReceiver::class.java)
    intent.putExtra("Service1", str)
    val pendingIntent =
        PendingIntent.getBroadcast(applicationContext, 234324243, intent, 0)
    val alarmManager = getSystemService(ALARM_SERVICE) as AlarmManager
    if(str == "Start") {
        alarmManager.setExact(
            AlarmManager.RTC_WAKEUP,
            millisTime,
            pendingIntent
        )
    }else if(str == "Stop")
    {
        alarmManager.cancel(pendingIntent)
    }
}
```

Play Alarm Sound in BroadcastReceiver

```
class AlarmBroadcastReceiver : BroadcastReceiver() {
    var mp: MediaPlayer? = null
    override fun onReceive(context: Context?, intent: Intent?) {
        var note:Notes? = null

        if(intent != null) {

                mp = MediaPlayer.create(context, R.raw.alarm);
                mp?.start()

        }
    }
```

Stop alarm

```
class AlarmBroadcastReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context?, intent: Intent?) {
        var note:Notes? = null
        if(intent != null && context!=null) {
            val str1 = intent.getStringExtra("Service1")
            if(str1 == null) {}            }
            else if(str1 == "Start" || str1 == "Stop"){
                val intentService = Intent(context, AlarmService::class.java)
                intentService.putExtra("Service1", intent.getStringExtra("Service1"))
                if(str1 == "Start")
                    context.startService(intentService)
                else if(str1=="Stop")
                    context.stopService(intentService)
            }
        }
    }
```

Stop Alarm

```
class AlarmService : Service() {
    var mp: MediaPlayer? = null
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        if(intent != null) {
            mp = MediaPlayer.create(this, R.raw.alarm);
            mp?.start()
        }
        return START_STICKY
    }

    override fun onDestroy() {
        mp?.stop()
        super.onDestroy()
    }
}
```

Read SMS

```
data class SMS(val sender: String, val message: String, val timestamp: Long)

fun readSMS(context: Context): List<SMS> {
    val smsList = ArrayList<SMS>()

    // Step 1: Check for READ_SMS permission
    if (ContextCompat.checkSelfPermission(
            context,
            android.Manifest.permission.READ_SMS
        ) != PackageManager.PERMISSION_GRANTED
    ) {
        // Permission not granted, return an empty list
        return smsList
    }
```

Read SMS Permission Request

```
private fun requestSMSPermission() {
    if (ActivityCompat.shouldShowRequestPermissionRationale(this, Manifest.permission.READ_SMS)) {
        // You may display a non-blocking explanation here, read more in the documentation:
        // https://developer.android.com/training/permissions/requesting.html
    }
    ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.READ_SMS,
        Manifest.permission.SEND_SMS,
        Manifest.permission.RECEIVE_SMS),
        SMS_PERMISSION_CODE)
}

private fun checkRequestPermission(): Boolean {
    return if (!isSMSReadPermission || !isSMSWritePermission) {
        requestSMSPermission()
        false
    } else true
}
```

Read SMS

```
    // Step 2: Read SMS data
    val uri = Uri.parse("content://sms/inbox")
    val contentResolver: ContentResolver = context.contentResolver
    val cursor: Cursor? = contentResolver.query(uri, null, null, null, null)

    cursor?.use {
        val senderIndex = it.getColumnIndex("address")
        val messageIndex = it.getColumnIndex("body")
        val timestampIndex = it.getColumnIndex("date")

        while (it.moveToNext()) {
            val sender = it.getString(senderIndex)
            val message = it.getString(messageIndex)
            val timestamp = it.getLong(timestampIndex)

            val sms = SMS(sender, message, timestamp)
            smsList.add(sms)
        }
    }
    return smsList
}
```

Send SMS

Ganpat University || विद्यया समाजोत्कर्षः || U.V. Patel College of Engineering

```
fun sendSMSMessage(context: Context, phoneNumber: String, message: String) {
    val SMS_PERMISSION_REQUEST = 899
    if (ContextCompat.checkSelfPermission(
            context,
            android.Manifest.permission.SEND_SMS
        ) == PackageManager.PERMISSION_GRANTED
    ) {
        // Permission is granted, send the SMS
        val smsManager =
            ContextCompat.getSystemService<SmsManager>(context, SmsManager::class.java)!!
                .createForSubscriptionId(SubscriptionManager.getDefaultSmsSubscriptionId())
        smsManager.sendTextMessage(phoneNumber, null, message, null, null)
        Toast.makeText(context, "SMS sent successfully", Toast.LENGTH_SHORT).show()
    } else {
        // Permission is not granted, request it
        ActivityCompat.requestPermissions(
            context as Activity,
            arrayOf(android.Manifest.permission.SEND_SMS),
            SMS_PERMISSION_REQUEST
        )
    }
}
```

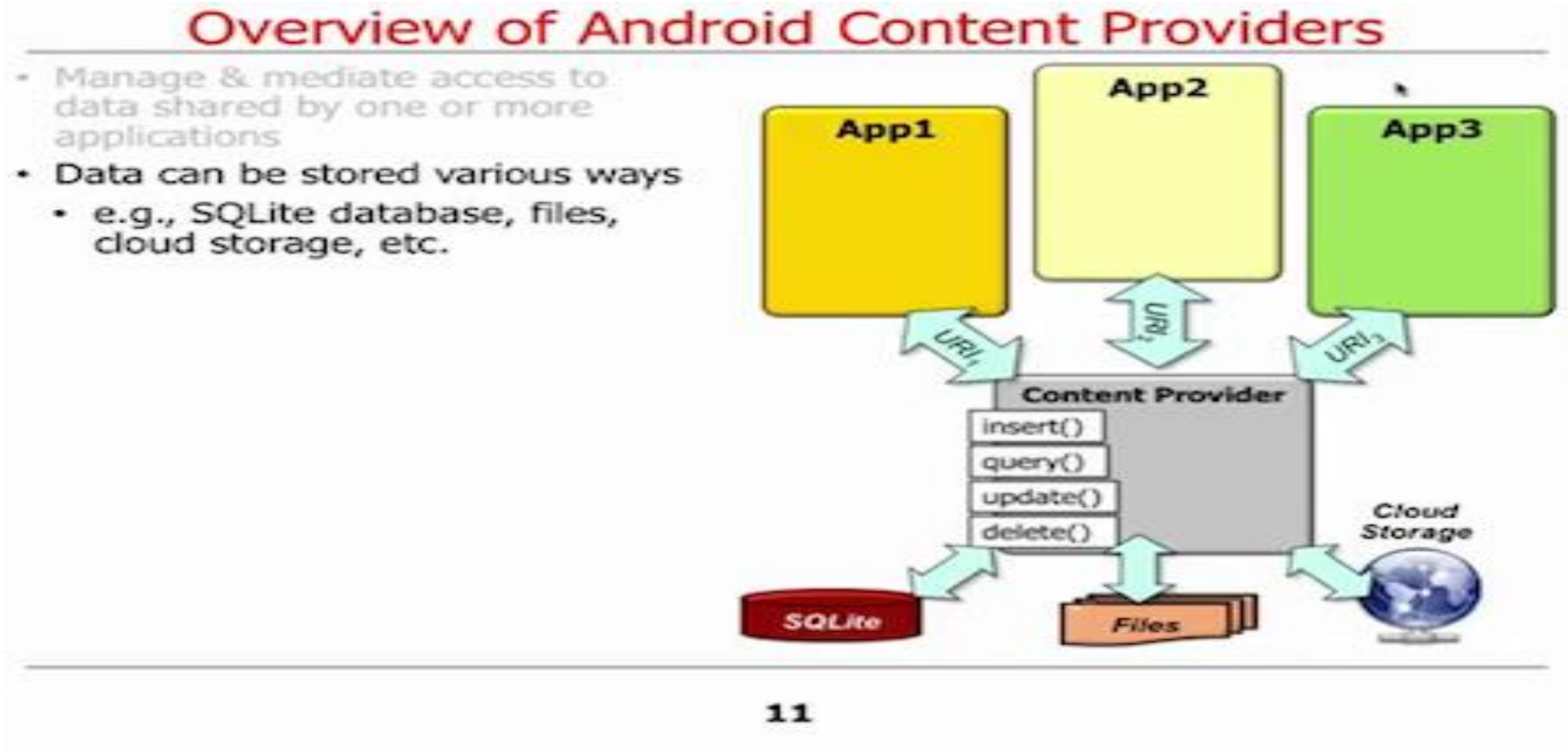
Receive SMS Broadcast Receiver

```xml
<receiver android:name=".SMSBroadcastReceiver"
    android:exported="true">
    <intent-filter>
        <action android:name="Telephony.Sms.Intents.SMS_RECEIVED_ACTION"/>
    </intent-filter>
</receiver>
```

```kotlin
class SMSBroadcastReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        if (intent.action == Telephony.Sms.Intents.SMS_RECEIVED_ACTION) {
            var sPhoneNo = ""
            var sSMSBody = ""
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                for (smsMessage in Telephony.Sms.Intents.getMessagesFromIntent(intent)) {
                    sPhoneNo = smsMessage.displayOriginatingAddress
                    sSMSBody += smsMessage.messageBody
                }
                //Actions
            }
        }
    }
}
```

Ganpat University
॥ विद्यया समाजोत्कर्षः ॥
U.V. Patel College of Engineering

Content Provider



Overview of Android Content Providers

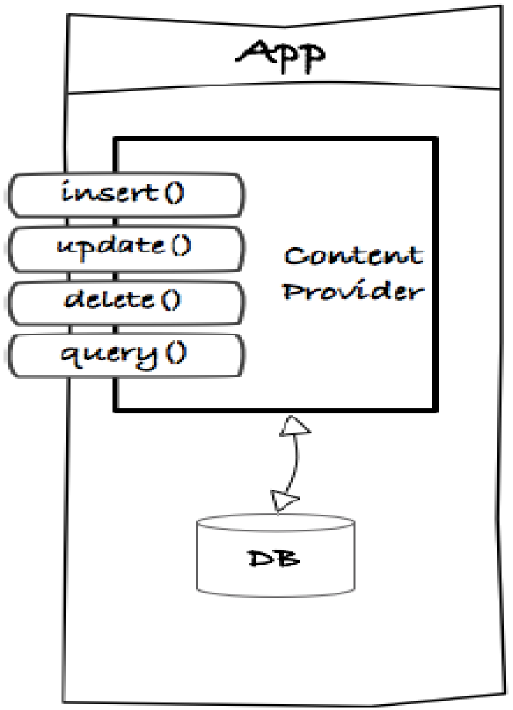
- Manage & mediate access to data shared by one or more applications
- Data can be stored various ways
  - e.g., SQLite database, files, cloud storage, etc.

App1
App2
App3
URI1
URI2
URI3
Content Provider
insert()
query()
update()
delete()
SQLite
Files
Cloud Storage

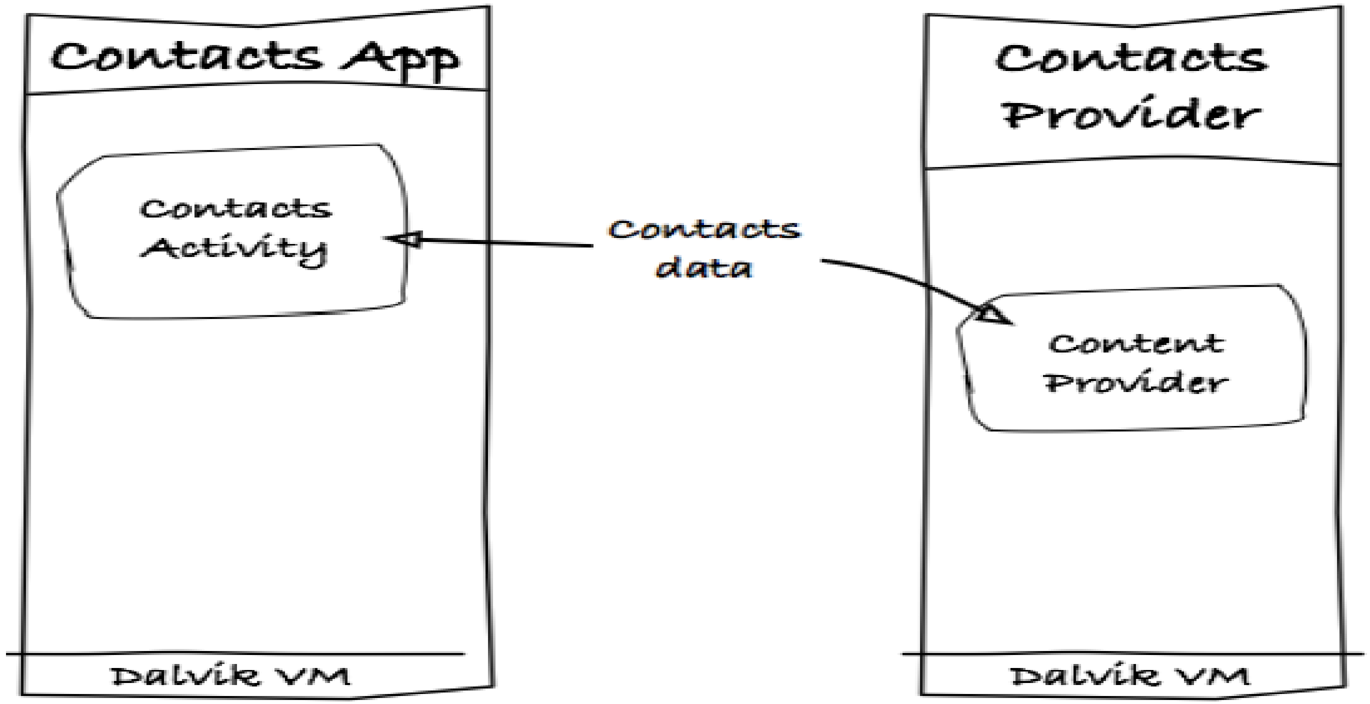
11

Content Provider

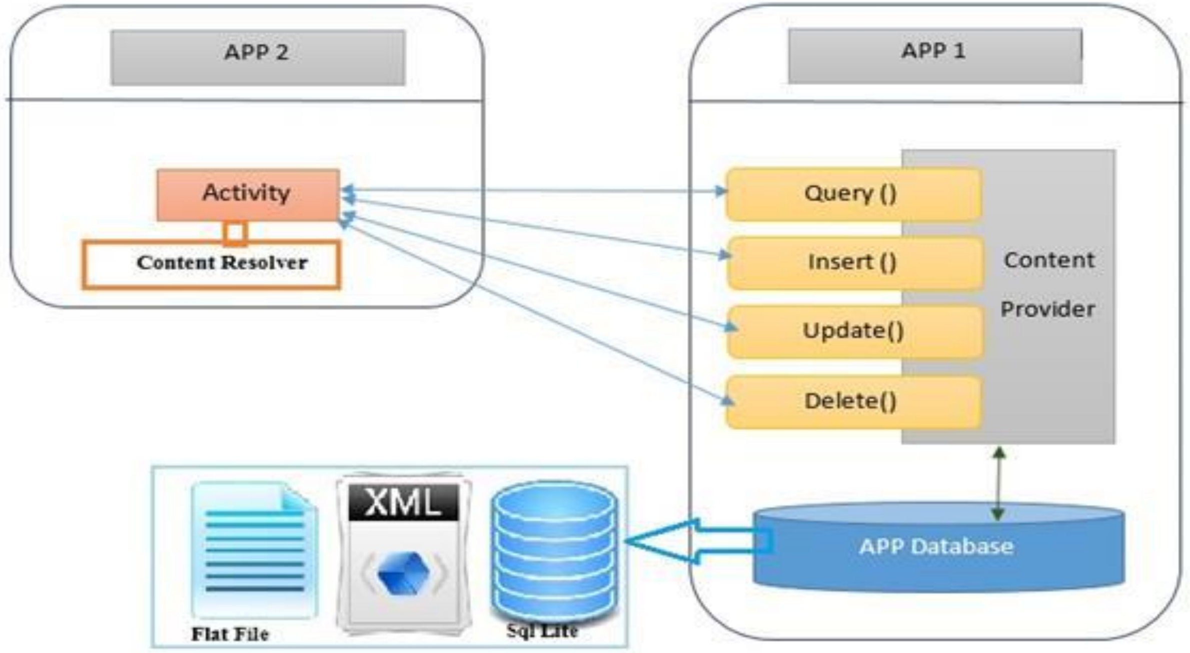
Content Providers share content with applications across application boundaries.
Examples of built-in Content Providers are: Contacts, MediaStore, Settings and more.



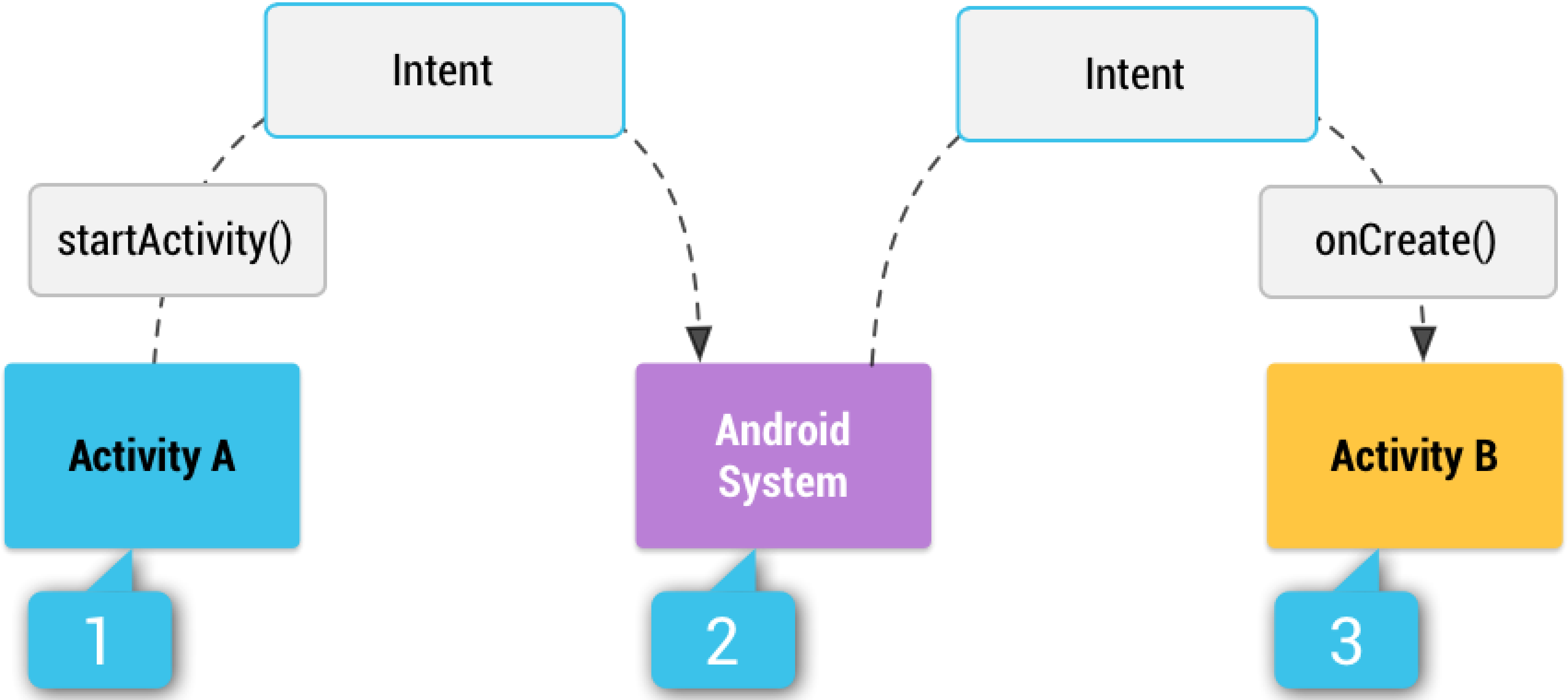
Content Provider



Content Provider



Intent


Intent
Intent
startActivity()
onCreate()
Activity A
Android System
Activity B
1
2
3

Intent

An Intent is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways

Intent

Starting an activity

startActivity().

Starting a service

startService()
bindService().

Delivering a broadcast

sendBroadcast()

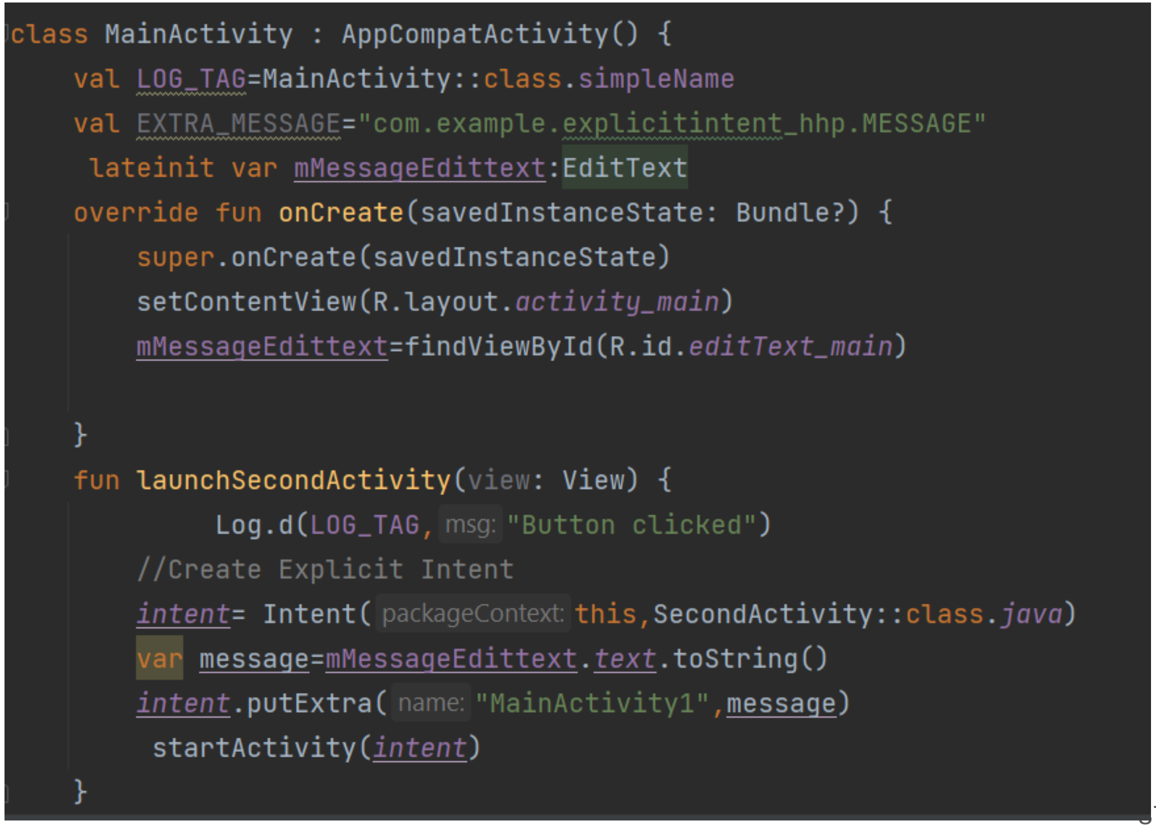
Intent Type

Explicit intents

Implicit intents

Intent Type

Explicit intent:-

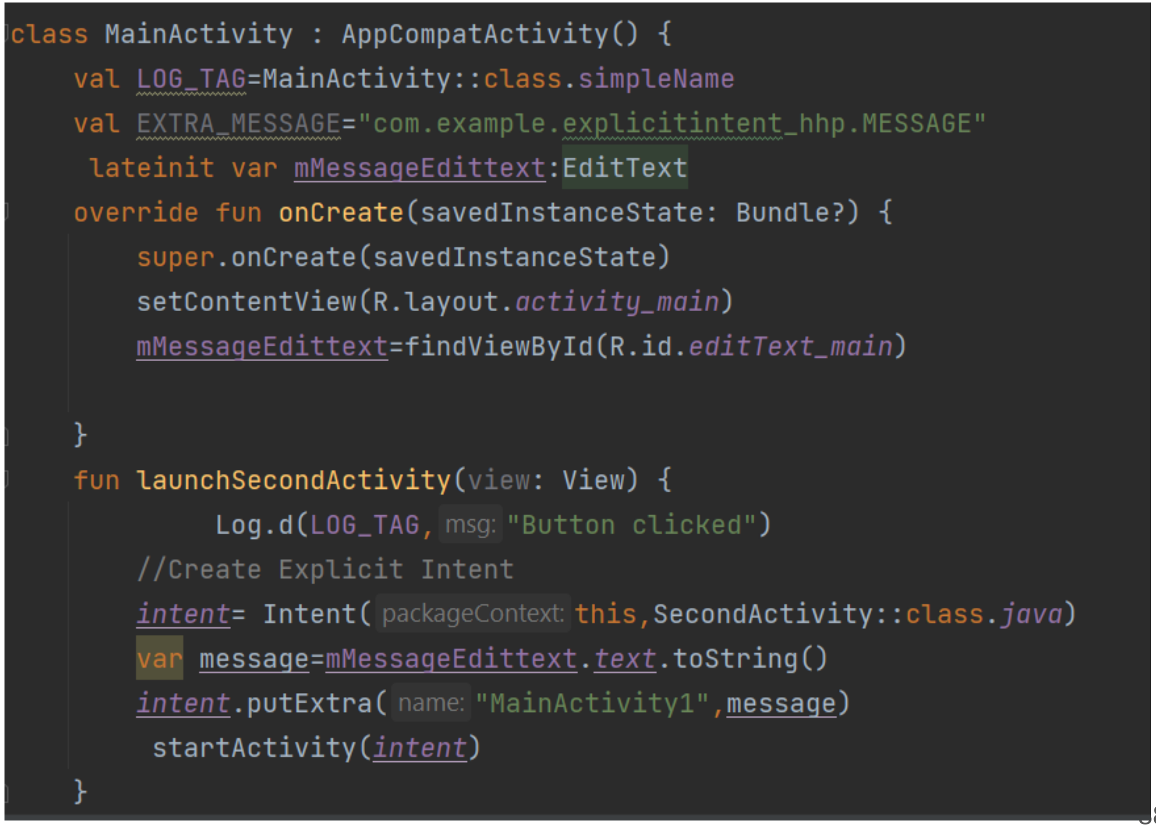


```
class MainActivity : AppCompatActivity() {
    val LOG_TAG=MainActivity::class.simpleName
    val EXTRA_MESSAGE="com.example.explicitintent_hhp.MESSAGE"
     lateinit var mMessageEdittext:EditText
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        mMessageEdittext=findViewById(R.id.editText_main)

    }
    fun launchSecondActivity(view: View) {
             Log.d(LOG_TAG, msg: "Button clicked")
        //Create Explicit Intent
        intent= Intent( packageContext: this,SecondActivity::class.java)
        var message=mMessageEdittext.text.toString()
        intent.putExtra( name: "MainActivity1",message)
         startActivity(intent)
    }
```

Intent Type

Explicit intent:-

MainActivity.kt

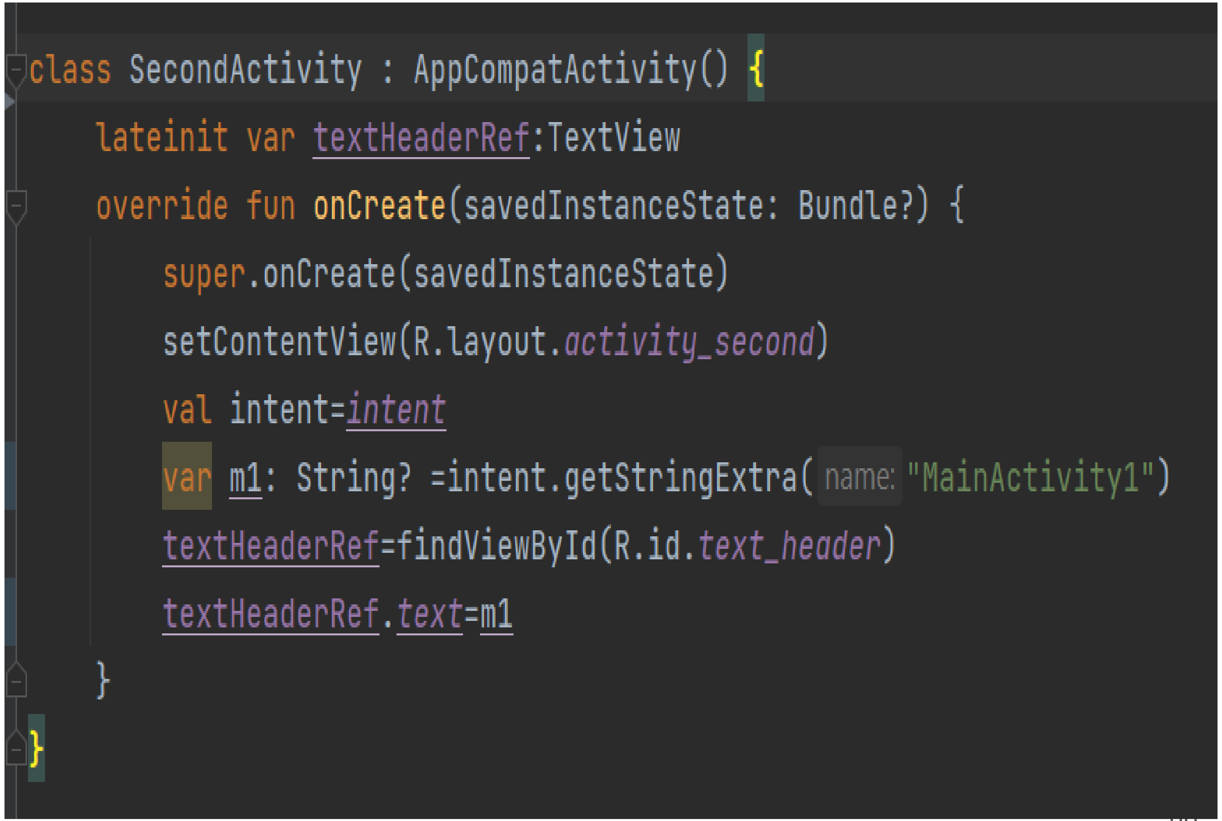


```
class MainActivity : AppCompatActivity() {
    val LOG_TAG=MainActivity::class.simpleName
    val EXTRA_MESSAGE="com.example.explicitintent_hhp.MESSAGE"
     lateinit var mMessageEdittext:EditText
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        mMessageEdittext=findViewById(R.id.editText_main)

    }
    fun launchSecondActivity(view: View) {
             Log.d(LOG_TAG, msg: "Button clicked")
        //Create Explicit Intent
        intent= Intent( packageContext: this,SecondActivity::class.java)
        var message=mMessageEdittext.text.toString()
        intent.putExtra( name: "MainActivity1",message)
         startActivity(intent)
    }
```

Intent Type

Explicit intent:-

SecondActivity.kt



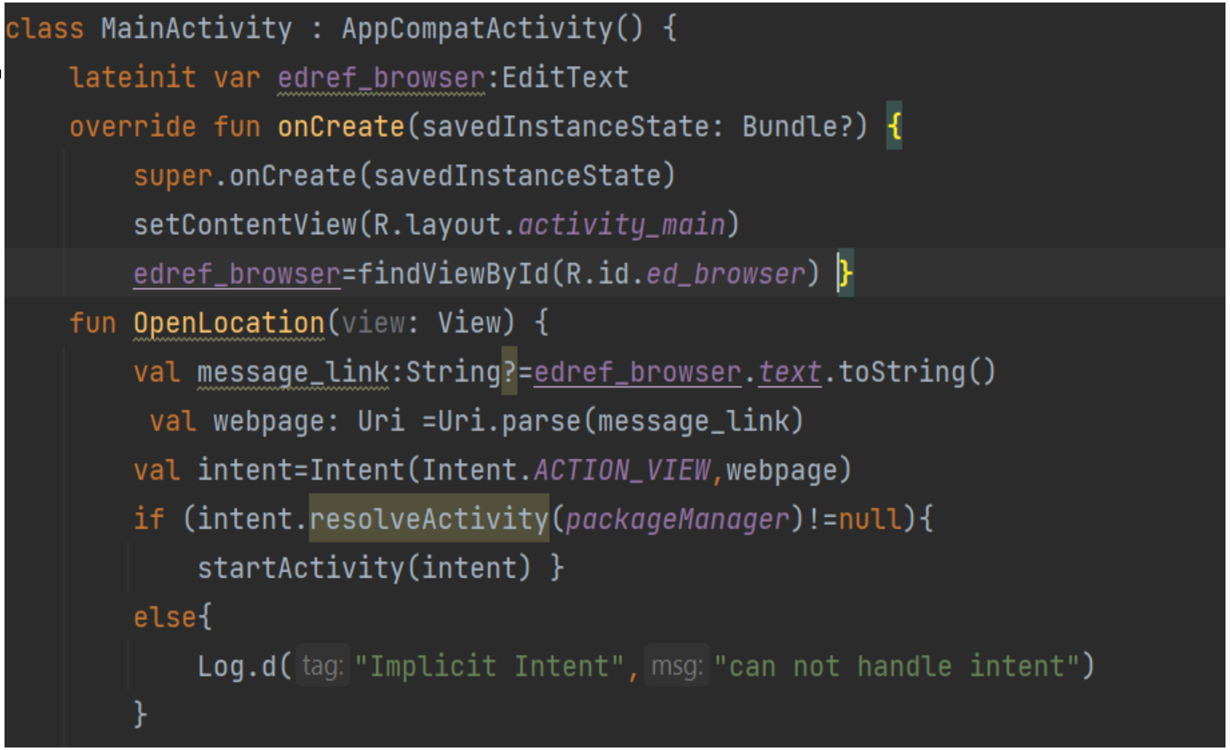
```
class SecondActivity : AppCompatActivity() {
    lateinit var textHeaderRef:TextView
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_second)
        val intent=intent
        var m1: String? =intent.getStringExtra( name: "MainActivity1")
        textHeaderRef=findViewById(R.id.text_header)
        textHeaderRef.text=m1
    }
}
```

Intent Type

Ganpat University || विद्यया समाजोत्कर्षः || U.V. Patel College of Engineering

Implicit intent:-

MainActivity.kt



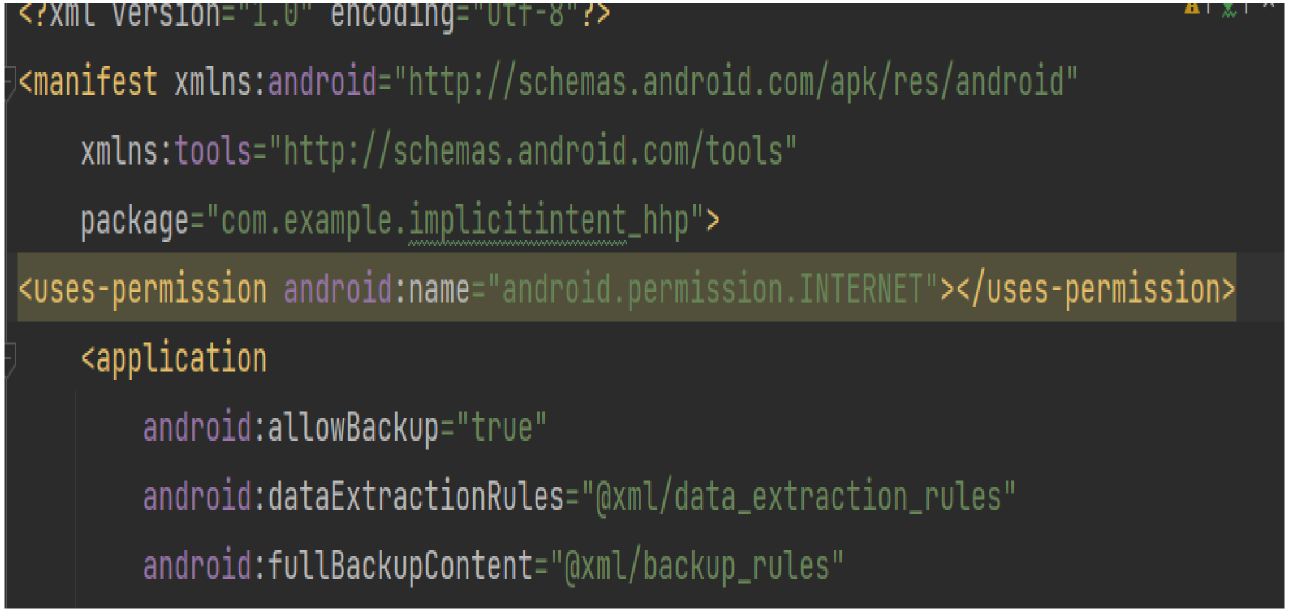
```
class MainActivity : AppCompatActivity() {
    lateinit var edref_browser:EditText
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        edref_browser=findViewById(R.id.ed_browser) }
    fun OpenLocation(view: View) {
        val message_link:String?=edref_browser.text.toString()
         val webpage: Uri =Uri.parse(message_link)
        val intent=Intent(Intent.ACTION_VIEW,webpage)
        if (intent.resolveActivity(packageManager)!=null){
            startActivity(intent) }
        else{
            Log.d( tag: "Implicit Intent", msg: "can not handle intent")
        }
```

Intent Type

Ganpat University
॥ विद्यया समाजोत्कर्षः ॥
U.V. Patel College of Engineering

Implicit intent:-

Manifest.xml



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.implicitintent_hhp">
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
```

Notifications

- A notification is a message you can display to the user outside of your application's normal UI.
- Android Toast class provides a handy way to show users alerts but problem is that these alerts are not persistent which means alert flashes on the screen for a few seconds and then disappears.
- When you tell the system to issue a notification, it first appears as an icon in the notification area.
- To see the details of the notification, the user opens the notification drawer.
- Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

Create and Send Notifications

- Step 1 - Create Notification Builder
- Step 2 - Setting Notification Properties
- Step 3 - Attach Actions
- This is an optional part and required if you want to attach an action with the notification. An action allows users to go directly from the notification to an Activity in your application, where they can look at one or more events or do further work.
- Step 4 - Issue the notification
- https://developer.android.com/guide/topics/ui/notifiers/notifications

Notification

```kotlin
private fun notificationDialog(context: Context, cls: Class<*>, title: String,
descr: String,note:Notes) {
    val notificationManager =
context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
    val NOTIFICATION_CHANNEL_ID = "hit"
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        val notificationChannel = NotificationChannel(
            NOTIFICATION_CHANNEL_ID,
            "LoginRegistration", NotificationManager.IMPORTANCE_HIGH
        )
        // Configure the notification channel.
        notificationChannel.description = "LoginRegistration description"
        notificationChannel.enableLights(true)
        notificationChannel.lightColor = Color.RED
        notificationChannel.vibrationPattern = longArrayOf(0, 1000, 500, 1000)
        notificationChannel.enableVibration(true)
        notificationManager.createNotificationChannel(notificationChannel)
    }
}
```

Notification

```
val notificationIntent = Intent(context, cls)
    notificationIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
     val pendingIntent = PendingIntent.getActivity(
        context, Notes.REMINDER_REQUEST_CODE, notificationIntent,
        PendingIntent.FLAG_UPDATE_CURRENT
    )
    val notificationBuilder = NotificationCompat.Builder(context,
NOTIFICATION_CHANNEL_ID)
    notificationBuilder.setAutoCancel(true)
        .setDefaults(Notification.DEFAULT_ALL)
        .setWhen(System.currentTimeMillis())
        .setSmallIcon(R.mipmap.ic_launcher)
        .setTicker("LoginRegistration-Hit")
         .setContentTitle(title)
        .setContentText(descr)
        .setContentInfo(descr + "Information")
        .setAutoCancel(true)
        .setContentIntent(pendingIntent)
    notificationManager.notify(1, notificationBuilder.build())
}
```

Android Manifest

- Android projects use a special **configuration file** called the Android manifest file
- The Android application manifest file is a specially formatted **XML** file that must accompany each Android application.
- This file contains important information about the application’s **identity**.

Android Manifest

- Every app project must have an **AndroidManifest.xml** file (with precisely that name) at the root of the project source set.
- The manifest file describes essential information about your app to the Android build tools, the Android operating system, and Google Play.
- The **components of the app**, which include all activities, services, broadcast receivers, and content providers.
- Each component must define basic **properties** such as the name of its Kotlin or Java class.
- It can also declare capabilities such as which **device configurations** it can handle, and intent filters that describe how the component can be started.

Android Manifest

Ganpat University | U.V. Patel College of Engineering
॥ विद्यया समाजोत्कर्षः ॥

- The **permissions** that the app needs in order to access protected parts of the system or other apps. It also declares any permissions that other apps must have if they want to access content from this app.
- The **hardware and software features** the app requires, which affects which devices can install the app from Google Play.
- If you're using Android Studio to build your app, the manifest file is created for you, and most of the essential manifest elements are added as you build your app (especially when using code templates).

App components

- For each app component that you create in your app, you must declare a corresponding XML element in the manifest file:
    - <activity> for each subclass of Activity.
    - <service> for each subclass of Service.
    - <receiver> for each subclass of BroadcastReceiver.
    - <provider> for each subclass of ContentProvider.

App components

- If you subclass any of these components without declaring it in the manifest file, the system cannot start it.

- The name of your subclass must be specified with the name attribute, using the full package designation. For example, an Activity subclass can be declared as follows:

```
<manifest ... >
    <application ... >
        <activity android:name="com.example.myapp.MainActivity" ... >
        </activity>
    </application>
</manifest>
```

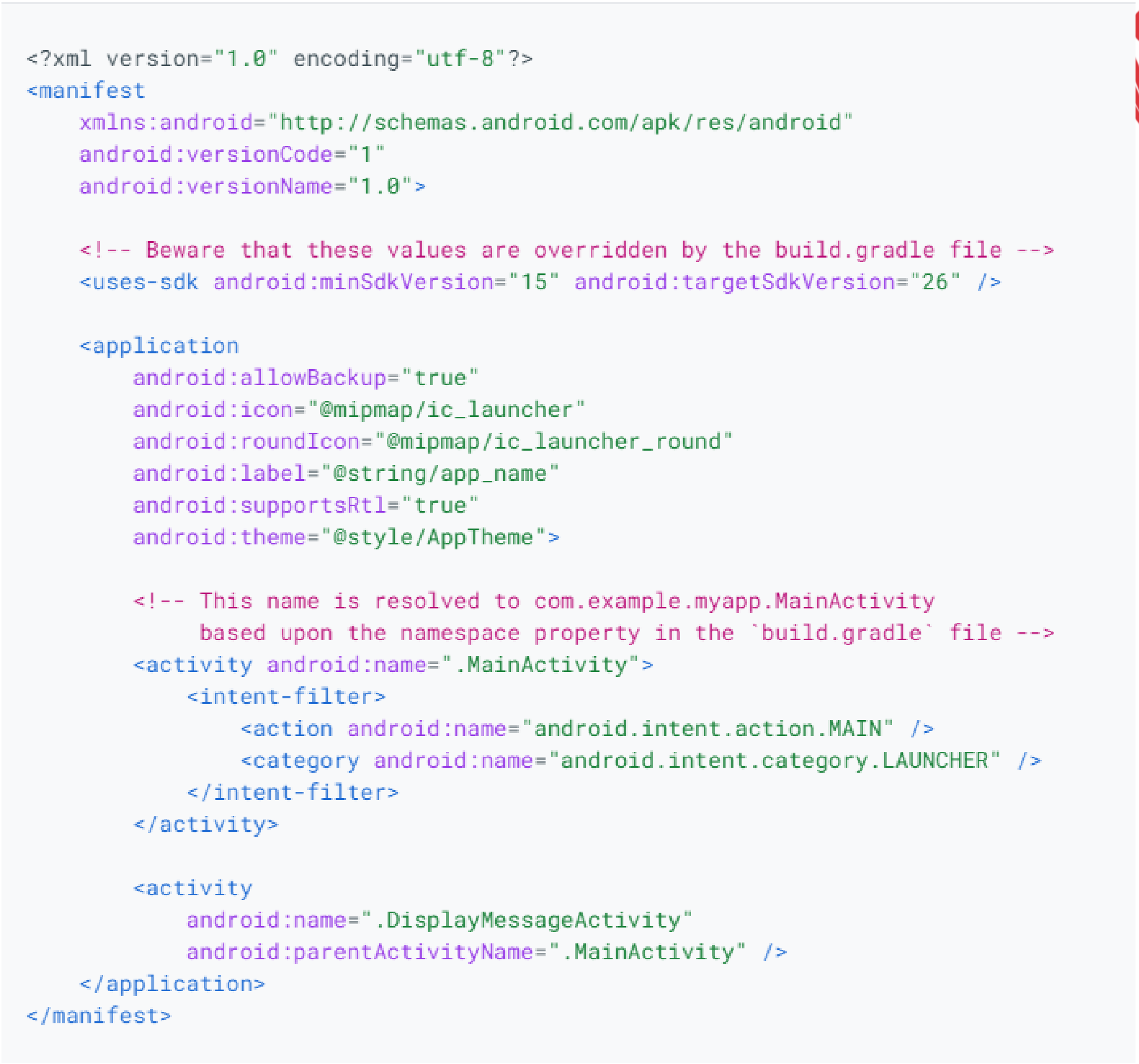
App components

- Intent filters
  - App activities, services, and broadcast receivers are activated by intents. An intent is a message defined by an Intent object that describes an action to perform, including the data to be acted upon, the category of component that should perform the action, and other instructions.
  - When an app issues an intent to the system, the system locates an app component that can handle the intent based on intent filter declarations in each app's manifest file.
- Icons and Labels
  - The icon and label that are set in the **<application>** element are the default icon and label for each of the app's components (such as all activities).

App components

- Permissions
- Android apps must request permission to access sensitive user data (such as contacts and SMS) or certain system features (such as the camera and internet access). Each permission is identified by a unique label.

```
<manifest ... >
    <uses-permission android:name="android.permission.SEND_SMS"/>
    ...
</manifest>
```



```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0">

    <!-- Beware that these values are overridden by the build.gradle file -->
    <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="26" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <!-- This name is resolved to com.example.myapp.MainActivity
             based upon the namespace property in the `build.gradle` file -->
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".DisplayMessageActivity"
            android:parentActivityName=".MainActivity" />
    </application>
</manifest>
```

Ganpat University
|| विद्यया समाजोत्कर्षः ||
U.V. Patel College of Engineering

Example of Manifest file