Native Mobile Applications

TalTech, Andres Käver, 2020-2021, Fall semester

Skype: akaver Email: akaver@itcollege.ee

Android app components

- Logging
- Context
- Activity
- Intent
- Service
- Broadcasts

Android - logging

- Full support in ide
- Use android.util.Log
- Verbose Log.v(), debug Log.d(), info Log.i(), warn Log.w(), error Log.e(), or “what a terrible Failure” Log.wtf()
- Deployed application should not contain logging code!
- Use BuildConfig.DEBUG flag for checking state (deployed or not)
- TAG – string

```
class MainActivity : AppCompatActivity() {
   // val TAG =  this::class.java.simpleName
   companion object {
      private val TAG = this::class.java.declaringClass!!.simpleName
   }
```

- public static int w (String tag, Throwable tr)
- public static int w (String tag, String msg, Throwable tr)

Android - Context

- The Context class is an “Interface to global information about an application environment".
- The Context class itself is declared as abstract class, whose implementation is provided by the Android OS. The documentation further provides that Context “…allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc".
- the Context provides the answer to the components question of “where the hell am I in relation to app/system generally and how do I access/communicate with the rest of the app?”

Android - activity

- Activity – one screen (UI and code)
- User interface – 1..n activities
- Every activity is separate component
- Activity can start other activities
- Back stack (lifo)
- Has to be declared in manifest also!

Android - activity

- Activity
- FragmentActivity
- ListActivity
- PreferenceActivity
- TabActivity

Android - activity

- One activity is designated as “Main”
  - Launched on first app activation
- Every time new activity is started, previous one is stopped
- Previous activity is stored in the back stack
- When activity is stopped/paused, callback methods are called
- Callbacks – create, resume, stop, destroy, etc…

Android – new activity

- Create subclass of Activity (or subclass of subclass of Activity)
- Implement callbacks
  - override fun OnCreate(...)
- Implement user interface
  - XML layout file
  - Or programmatically

Android – new activity, manifest

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

- Declaration in AndroidManifest is mandatory
- Specify intent filters
  - Intent filter declares, how **other** system components may use this activity
- Auto-created stub for main activity
  - Action action.MAIN – activity responds to the “main” action
  - Category category.LAUNCHER – actitvity is placed into launcher category

```
<activity android:name=".MainActivity" android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Android – new activity, starting

- startActivity(intent)
- Starting your own activity – specify class name

```
val intent = Intent(this, OtherActivity::class.java)
startActivity(intent)
```

- Calling other activities

```
val intent = Intent(Intent.ACTION_SEND)
intent.putExtra(Intent.EXTRA_EMAIL, recipientArray)
startActivity(intent)
```

- Intent.EXTRA_EMAIL – stores list of email recipients

Android – new activity, starting for result

- StartActivityForResult()
- Implement onActivityResult() callback method

```
val PICK_CONTACT_REQUEST = 1234

fun pickContact(){
   val intent = Intent(Intent.ACTION_PICK, ContactsContract.Contacts.CONTENT_URI)
   startActivityForResult(intent, PICK_CONTACT_REQUEST)
}

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
  if (resultCode == Activity.RESULT_OK && requestCode == PICK_CONTACT_REQUEST){
     val cursor = contentResolver.query(data!!.data!!,arrayOf(ContactsContract.Contacts.DISPLAY_NAME) ,null,null,null )
     if (cursor!!.moveToFirst()){
        val columnIndex = cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME)
        val name = cursor.getString(columnIndex)
        Log.d(TAG, "Name: " + name)
     }
  }
}
```

Android – shut down activity

- finish() – activity closes itself
- shut down previously started activity – finishActivity(id)

Android – activity lifecycle

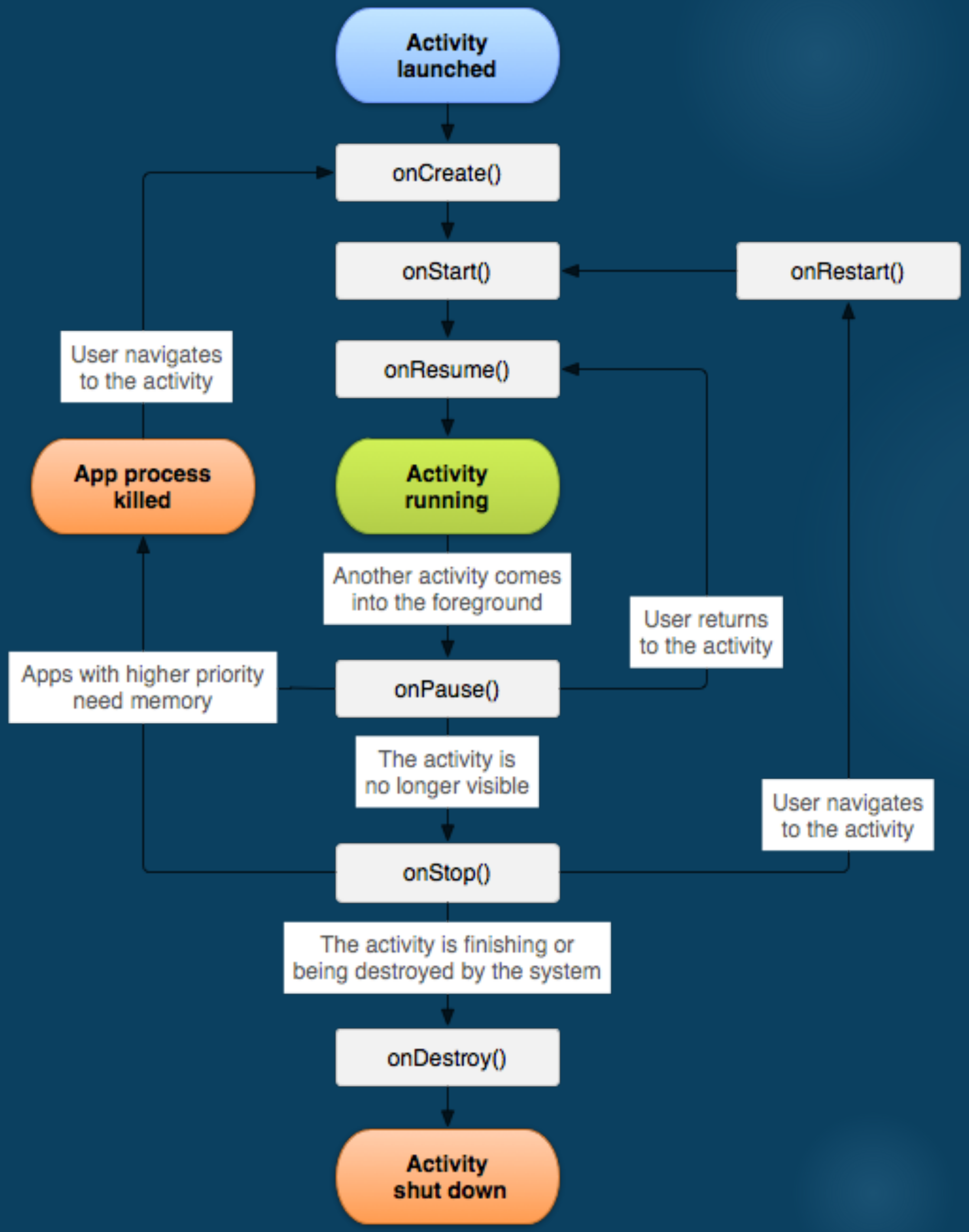
- Three essential states
  - Resumed
    - In foreground, has user focus. “running”
  - Paused
    - Activity is partially visible, and is “alive”. Can be killed by system in low memory situation
  - Stopped
    - Activity is 100% obscured by another activity. It is alive, but is not attached to the window manager. Can be killed by system, when memory is needed.
- Paused or Stopped – system calls finish() method on activity. When activity is reopened, it must be created again

Android – Lifecycle callbacks

- Fundamental callbacks
- Must always call the superclass implementation before doing any work

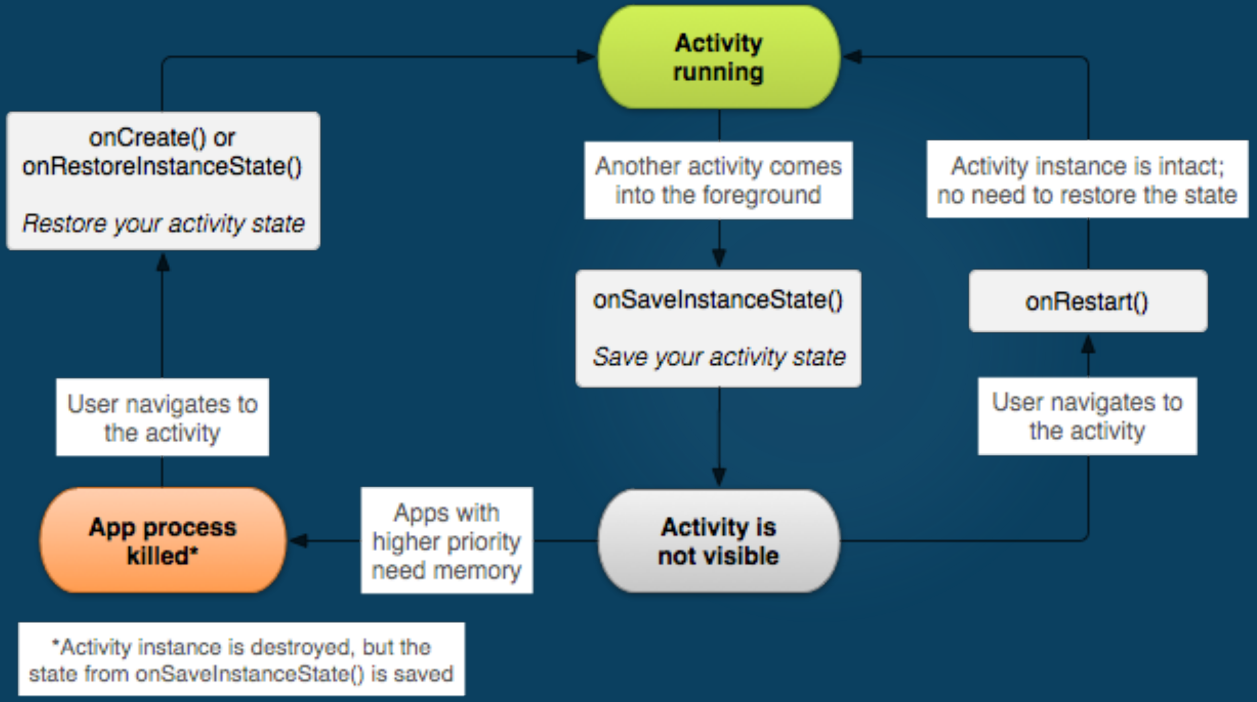
```
class MainActivity : AppCompatActivity() {
   val TAG =  this::class.java.simpleName

   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      // The activity is being created.
      setContentView(R.layout.activity_main)
   }

   override fun onStart() {
      super.onStart()
      // The activity is about to become visible.

   }

   override fun onResume() {
      super.onResume()
      // The activity has become visible (it is now "resumed").
   }

   override fun onPause() {
      super.onPause()
      // Another activity is taking focus (this activity is about to be "paused").
   }

   override fun onStop() {
      super.onStop()
      // The activity is no longer visible (it is now "stopped")
   }

   override fun onDestroy() {
      super.onDestroy()
      // The activity is about to be destroyed.

   }
```

Android – Lifecycle


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 - SaveInstanceState

- system calls onSaveInstanceState() before making the activity vulnerable to destruction
- Passes Bundle, as name-value pairs
- Save state, using
  - putString() and putInt()
- Bundle is passed back in
  - onCreate() and onRestoreInstanceState()


Activity running
onCreate() or onRestoreInstanceState()
Restore your activity state
Another activity comes into the foreground
Activity instance is intact; no need to restore the state
onSaveInstanceState()
Save your activity state
onRestart()
User navigates to the activity
User navigates to the activity
App process killed*
Apps with higher priority need memory
Activity is not visible
*Activity instance is destroyed, but the state from onSaveInstanceState() is saved

Android - handling conf changes

- Orientation change, physical keyboard, language
- System recreates the running activity
  - calls onDestroy(),
  - then immediately calls onCreate()

Android – Intent and Intent filters

- Messaging object, used for requesting action from another app component
- Fundamental uses
  - Start an activity
    - startActivity or startActivityForResult
  - Start a service
    - startService or bindService
  - Deliver broadcast
    - sendBroadcast, sendOrderedBroadcast, or sendStickyBroadcast

Android – Intent and Intent filters

- Explicit intents
  - Specify component by name (usually in your own app)
- Implicit intents
  - Declare general action to perform
  - System searches in manifests (intent-filter) for suitable activity
  - If several are found, user is presented with dialog for picking

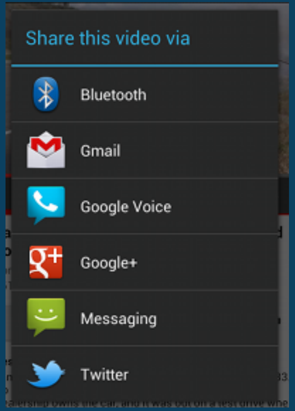
Android - intents

```
fun startWithExplicitIntent(){
    val intent = Intent(this, MyService::class.java)
    startService(intent)
}
```

```
fun sendTextMessage(){
    // Create the text message with a string
    val sendIntent = Intent(Intent.ACTION_SEND)
    sendIntent.putExtra(Intent.EXTRA_TEXT, "Hello!")
    sendIntent.type = "text/plain"

    // Verify that the intent will resolve to an activity
    if (sendIntent.resolveActivity(packageManager) != null){
        startActivity(sendIntent)
    }
}
```

- Explicit intent
- Implicit intent
- Forcing an app chooser
  - To show the chooser, create an Intent using createChooser() and pass it to startActivity()

```
fun sendTextMessageAppChooser(){
    // Create the text message with a string
    val sendIntent = Intent(Intent.ACTION_SEND)
    sendIntent.putExtra(Intent.EXTRA_TEXT, "Hello!")
    sendIntent.type = "text/plain"

    val chooserIntent = Intent.createChooser(sendIntent, "Send via")
    // Verify that the intent will resolve to an activity
    if (sendIntent.resolveActivity(packageManager) != null){
        startActivity(chooserIntent)
    }
}
```



Android – intent filter

- To advertise which implicit intents your app can receive
- declare one or more intent filters for each of your app components with an <intent-filter> element in your manifest file
- Action
  - intent action accepted, in the name attribute
- Data
  - type of data accepted, using one or more attributes that specify various aspects of the data URI (scheme, host, port, path, etc.) and MIME type
- Category
  - category accepted, in the name attribute.
  - In order to receive implicit intents, you must include the CATEGORY_DEFAULT

Android – intent filter

- Activity declaration with an intent filter to receive an ACTION_SEND intent when the data type is text

```xml
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

- An implicit intent is tested against a filter by comparing the intent to each of the three elements.
- To be delivered to the component, the intent must pass all three tests.

Android – intent filter

- Social app

```
<activity android:name="MainActivity">
    <!-- This activity is the main entry, should appear in app launcher -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity android:name="ShareActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name="android.intent.action.SEND_MULTIPLE"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="application/vnd.google.panorama360+jpg"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="video/*"/>
    </intent-filter>
</activity>
```

Android – Intent

- Parse and check intent in OnCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    // The activity is being created.

    setContentView(R.layout.activity_main)

    if (intent.action == Intent.ACTION_SEND) {
        if (intent.type == "text/plain") {
            val text = intent.getStringExtra(Intent.EXTRA_TEXT)
            Log.d(TAG, text)
        }
    }
}
```

Android - Service

- Runs in the background without direct interaction with the user
- Not bound to the lifecycle of an activity
- Used for repetitive and potentially long running operations
  - Internet downloads
  - checking for new data
  - Streaming
  - GPS
- Service runs in the same process as the main thread of the app
- Use asynchronous processing in the service

Android – Service (platform)

- Predefined system services
- Application can use them, given the right permissions
  - getSystemService()

Android – Service (custom)

```
<service
    android:name="MyService"
    android:icon="@drawable/icon"
    android:label="@string/service_name"></service>
```

- Declare in manifest
  - Inside <application> tags!
- Extend the Service class or one of its subclasses.
- Start service
- Can also start via bindService(). Allows direct communication with the service
- Use android:exported="false“ for keeping service private

```
package ee.taltech.akaver.helloworld01

import android.app.Service
import android.content.Intent
import android.os.IBinder

class MyService: Service() {
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        // TODO: do something useful

        return START_NOT_STICKY
    }

    override fun onBind(intent: Intent?): IBinder? {
        return null
    }

}
```

```
fun startService(){
    val intent = Intent(this, MyService::class.java)
    intent.putExtra("KEY1", "Value1")
    startService(intent)
}
```

Android – service restart

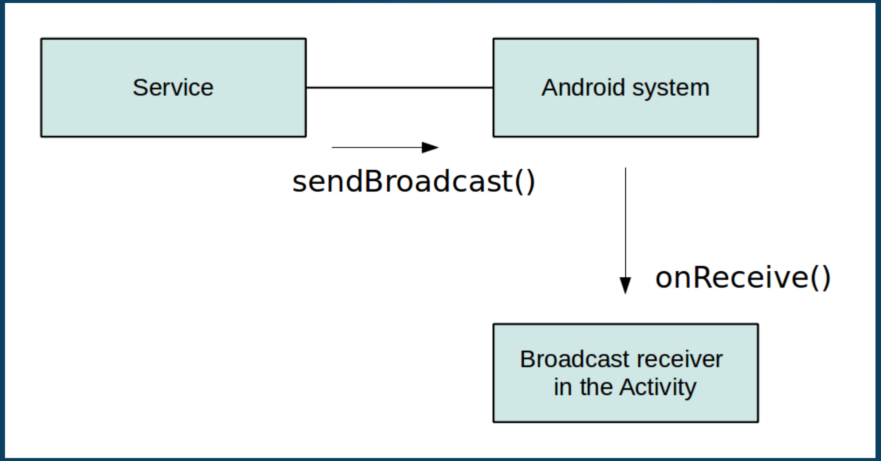
- START_STICKY
  - Service is restarted if it gets terminated. Intent data passed to the onStartCommand method is null. Used for services which manages their own state and do not depend on the Intent data.
- START_NOT_STICKY
  - Service is not restarted. Used for services which are periodically triggered anyway.
- START_REDELIVER_INTENT
  - Similar to Service.START_STICKY but the original Intent is re-delivered to the onStartCommand method.

Android – service stop

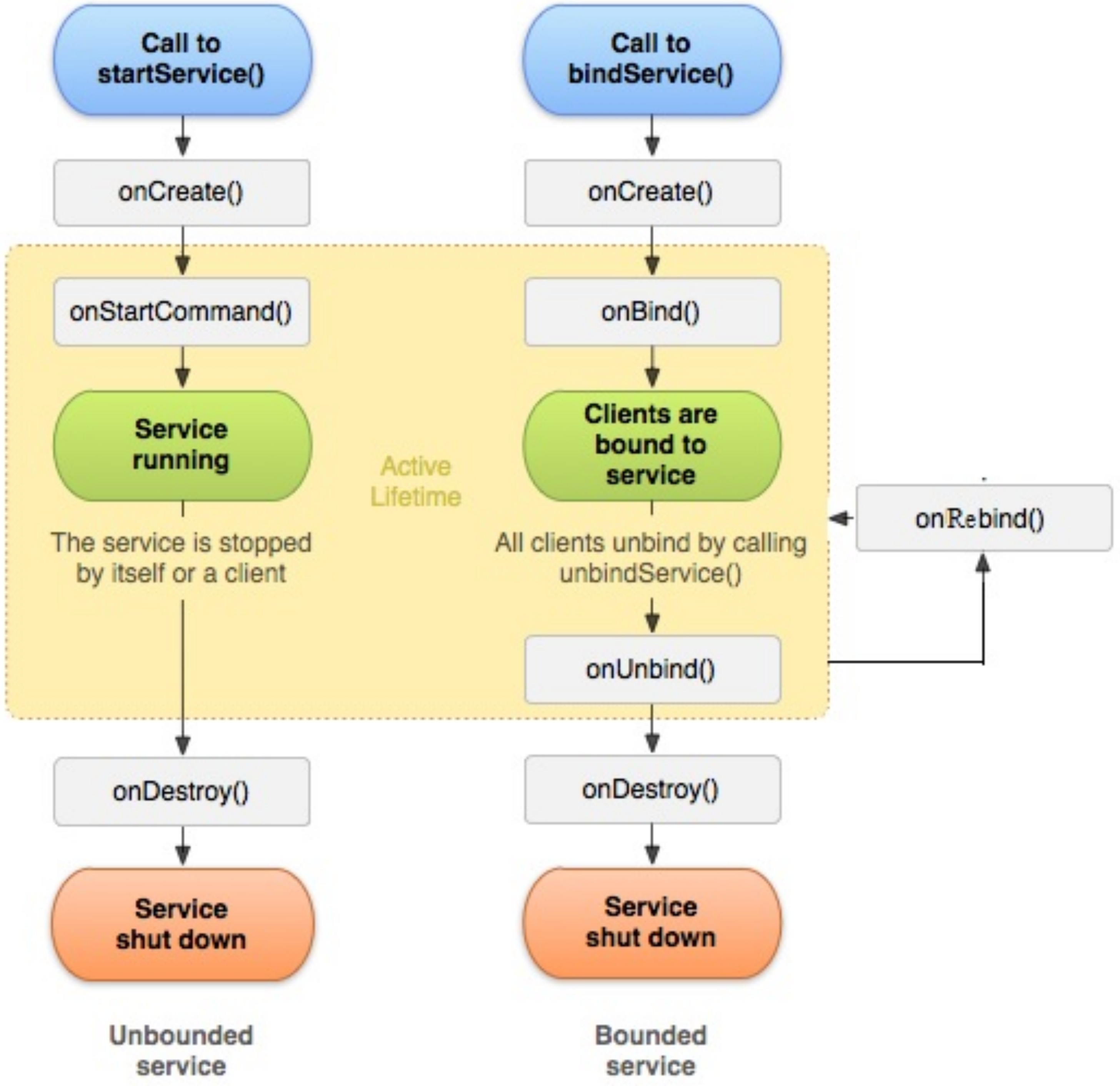
- stopService()
  - One call to the stopService() method stops the service.
- stopSelf() – service terminates itself. Used when service finishes its work.

Android – communication with service

- Simple scenario – no direct communication. Service receives intent when starting.
- Using receiver
  - Service broadcasts events
  - Activity registers broadcast receiver and receives events from service
- Activity binds to local service
  - IBinder, onBind()



Android service lifecycle



Android – Broadcasts

- Two types of messages – Local (inside your App) or Global
- IntentFilter
- Receiver declared in code or in Manifest

Android - Broadcasts

- IntentFilter – you can only receive declared broadcasts

```
private val localReceiverIntentFilter = IntentFilter()

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    // whitelist the actions we want to receive
    localReceiverIntentFilter.addAction(C.ACTION_TIMEMESSAGE);
    localReceiverIntentFilter.addAction(C.ACTION_AUDIO_PLAYING);
    localReceiverIntentFilter.addAction(C.ACTION_AUDIO_BUFFERING);
    localReceiverIntentFilter.addAction(C.ACTION_AUDIO_STOPPED);
}
```

Android - Broadcasts

- Internal broadcast receiver – inside main class (or service)

```
private inner class BroadcastReceiverInMainActivity: BroadcastReceiver(){
    override fun onReceive(context: Context?, intent: Intent?) {
        Log.d(TAG, "BroadcastReceiverInMainActivity.onReceive " + (intent?.action ?: "null intent"))
        when (intent?.action){
            C.ACTION_AUDIO_BUFFERING -> buttonPlayStop.text = "BUFFERING"
            C.ACTION_AUDIO_STOPPED -> buttonPlayStop.text = "PLAY"
            C.ACTION_AUDIO_PLAYING -> {
                audioStatus = C.AUDIO_PLAYING
                buttonPlayStop.text = "STOP"
            }
        }
    }
}
```

Android - Broadcasts

- Register and unregister your receiver
- Do not double register!

```
private val localReceiver = BroadcastReceiverInMainActivity()

override fun onResume() {
    super.onResume()
    LocalBroadcastManager
        .getInstance(this)
        .registerReceiver(localReceiver, localReceiverIntentFilter)
}

override fun onPause() {
    super.onPause()
    LocalBroadcastManager
        .getInstance(this)
        .unregisterReceiver(localReceiver)
}
```

Android – Broadcasts

- Send out local brodcasts (in service for example)

```
fun sendBroadcast(){
    // shout out the corridor door
    LocalBroadcastManager.
        getInstance(applicationContext).
        sendBroadcast( Intent(C.ACTION_AUDIO_BUFFERING))
}
```

Android

- The END!