

11. The Activity lifecycle


Apps, memory, and storage

- **storage:** Your device has apps and files installed and stored on its internal disk, SD card, etc.
 - **Settings → Storage**
- **memory:** Some subset of apps might be currently loaded into the device's RAM and are either running or ready to be run.
 - When the user loads an app, it is loaded from storage into memory.
 - When the user exits an app, it might be cleared from memory, or might remain in memory so you can go back to it later.
 - See which apps are in memory:
 - **Settings → Apps → Running**



3G

19:20

<  Appl.. Show cached processes

DOWNLOADED

RUNNING

ALL

Facebook

27MB

1 process and 1 service

20:17:47

WhatsApp

18MB

2 processes and 1 service

76:39:39

Messenger

18MB

1 process and 1 service

03:18

ConvertIt!

2.8MB

1 process and 1 service

76:39:37

Clipper

2.5MB

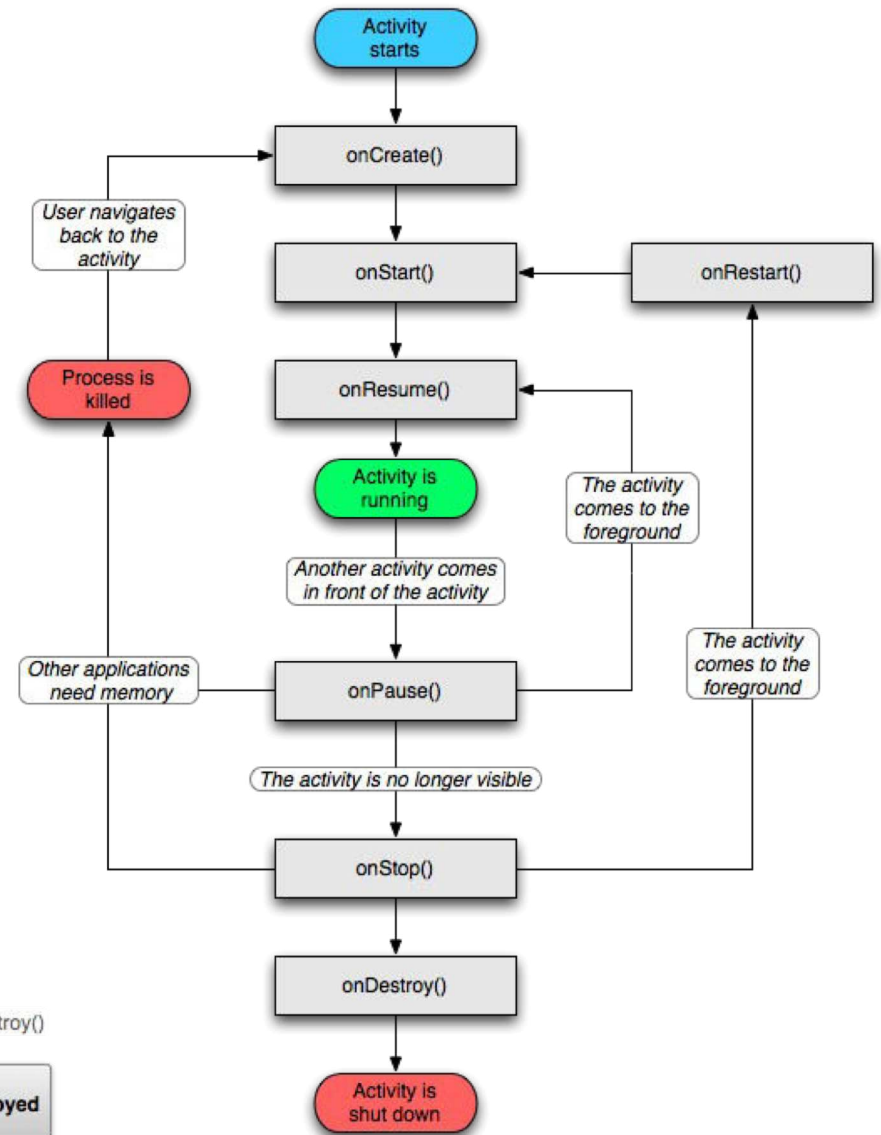
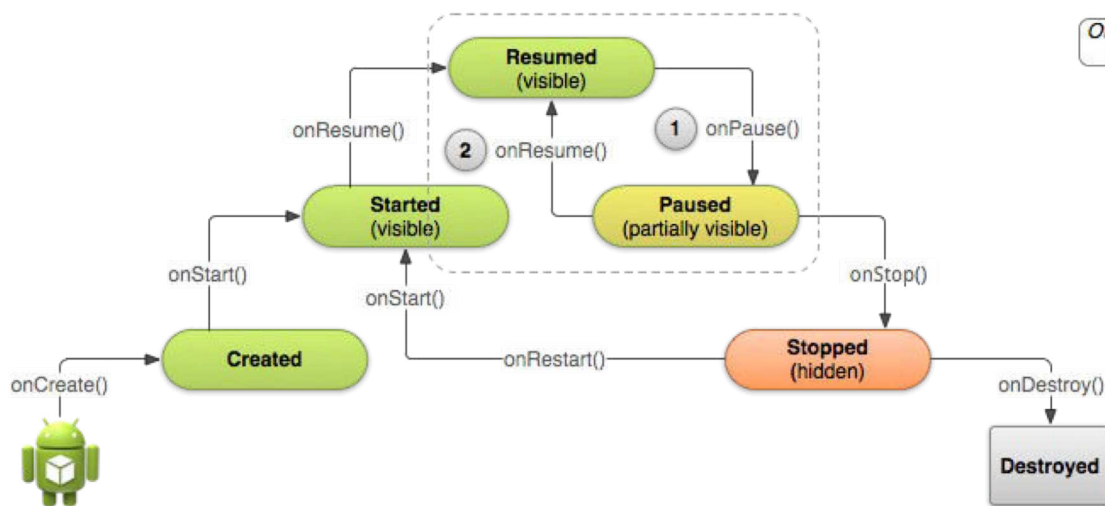
1 process and 1 service

76:39:30

Activity state

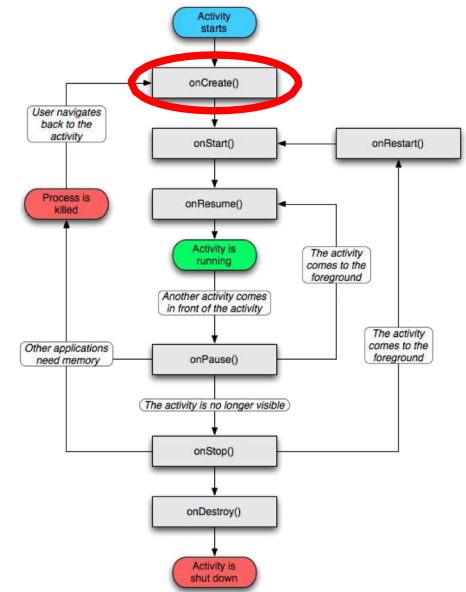
- An activity can be thought of as being in one of several states:
 - **starting**: In process of loading up, but not fully loaded.
 - **running**: Done loading and now visible on the screen.
 - **paused**: Partially obscured or out of focus, but not shut down.
 - **stopped**: No longer active, but still in the device's active memory.
 - **destroyed**: Shut down and no longer currently loaded in memory.
- Transitions between these states are represented by **events** that you can listen to in your activity code.
 - onCreate, onPause, onResume, onStop, onDestroy, ...

Other diagrams



The onCreate method

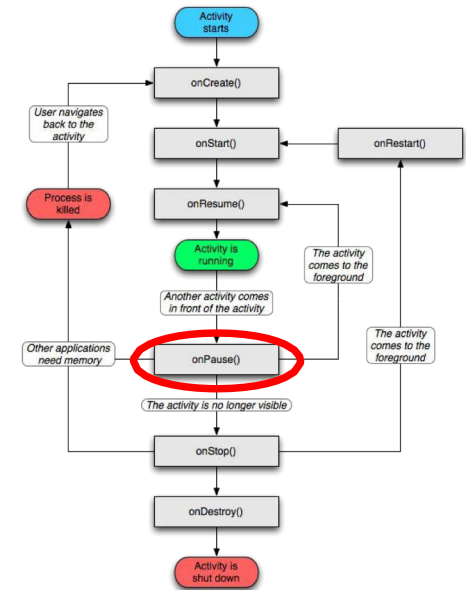
- In **onCreate**, you create and set up the activity object, load any static resources like images, layouts, set up menus etc.
 - after this, the Activity object exists
 - think of this as the "constructor" of the activity



```
public class FooActivity extends Activity {
    ...
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);    // always call super
        setContentView(R.layout.activity_foo); // setup layout
        anyotherinitialization code; // anythingelse you need
    }
}
```

The onPause method

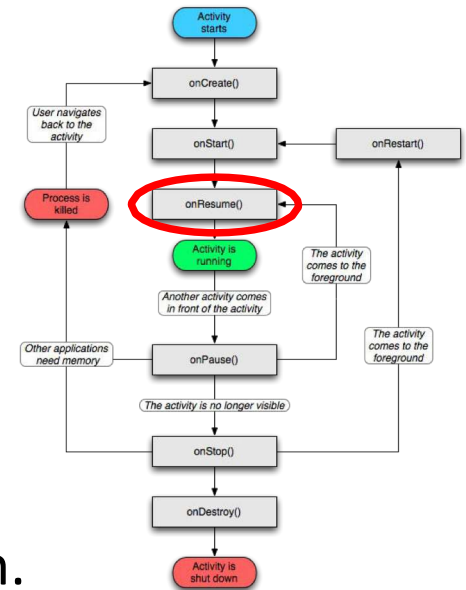
- When **onPause** is called, your activity is still partially visible.
- May be temporary, or on way to termination.
 - **Stop animations** or other actions that consume CPU.
 - **Commit unsaved changes** (e.g. draft email).
 - **Release system resources** that affect battery life.



```
public void onPause() {
    super.onPause();                // always call super
    if (myConnection != null) {
        myConnection.close();      // release resources
        myConnection = null;
    }
}
```

The onResume method

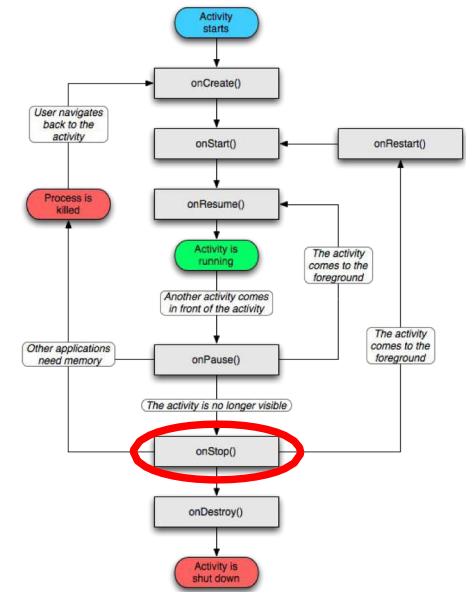
- When **onResume** is called, your activity is coming out of the Paused state and into the Running state again.
- Also called when activity is first created/loaded!
 - **Initialize resources** that you will release in onPause.
 - **Start/resume animations** or other ongoing actions that should only run when activity is visible on screen.



```
public void onResume() {
    super.onPause(); // always call super
    if (myConnection == null) {
        myConnection = new ExampleConnect(); // init.resources
        myConnection.connect();
    }
}
```

The onStop method

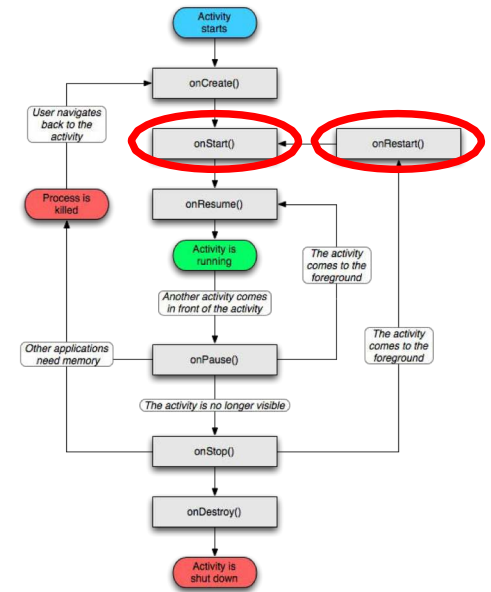
- When **onStop** is called, your activity is no longer visible on the screen:
 - User chose another app from **Recent Apps** window.
 - User starts a **different activity** in your app.
 - User receives a **phone call** while in your app.
- Your app might still be running, but that activity is not.
 - onPause is always called before onStop.
 - onStop performs heavy-duty shutdown tasks like writing to a database.



```
public void onStop() {
    super.onStop();           // always call super
    ...
}
```


onStart and onRestart

- **onStart** is called every time the activity begins.
- **onRestart** is called when activity *was* stopped but is started again later (all but the first start).
 - Not as commonly used; favor onResume.
 - Re-open any resources that onStop closed.



```
public void onStart() {  
    super.onStart();
```

// always call super

```
    ...
```

```
}
```

```
public void onRestart() {  
    super.onRestart();
```

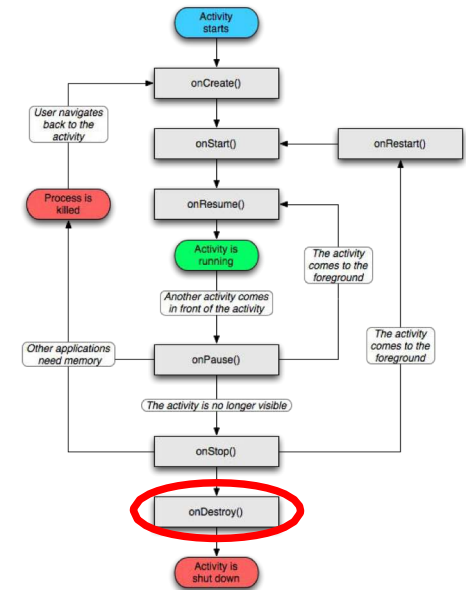
// always call super

```
    ...
```

```
}
```

The onDestroy method

- When **onDestroy** is called, your entire app is being shut down and unloaded from memory.
 - Unpredictable exactly when/if it will be called.
 - Can be called whenever the system wants to reclaim the memory used by your app.
 - Generally favor onPause or onStop because they are called in a predictable and timely manner.

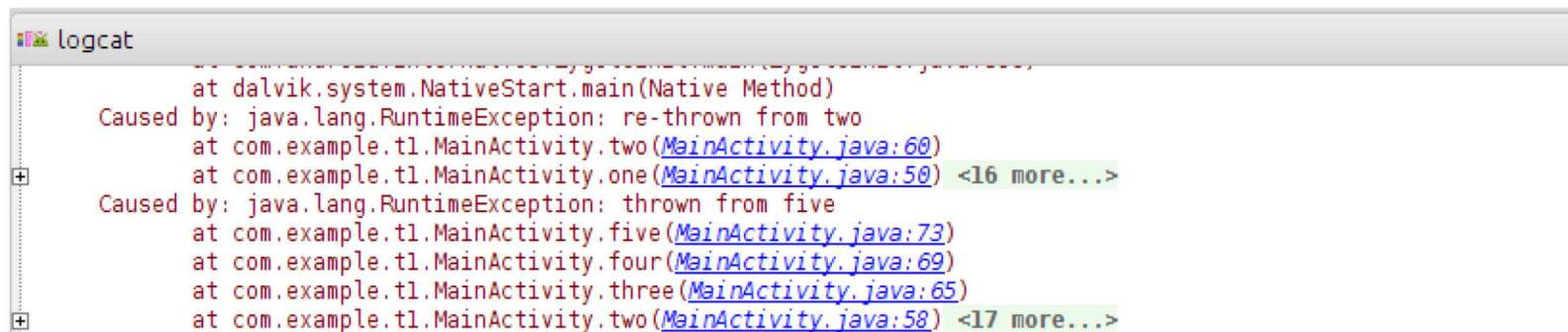


```
public void onDestroy() {
    super.onDestroy();           // always call super
    ...
}
```

Testing activity states

- Use the LogCat system for logging messages when your app changes states:
 - analogous to System.out.println debugging for Android apps
 - appears in the LogCat console in Android Studio

```
public void onStart() {  
    super.onStart();  
    Log.v("testing", "onStart was called!");  
}
```



Log methods

Method	Description
<code>Log.d("tag", "message");</code>	debug message (for debugging)
<code>Log.e("tag", "message");</code>	error message (fatal error)
<code>Log.i("tag", "message");</code>	info message (low-urgency FYI)
<code>Log.v("tag", "message");</code>	verbose message (rarely shown)
<code>Log.w("tag", "message");</code>	warning message (non-fatal error)
<code>Log.wtf("tag", <i>exception</i>);</code>	log stack trace of an exception

- Each method can also accept an optional exception argument:

```
try { someCode(); }  
catch (Exception ex) {  
    Log.e("error4", "something went wrong", ex);  
}
```