Mobile Development
Lecture 6: The Activity Lifecycle

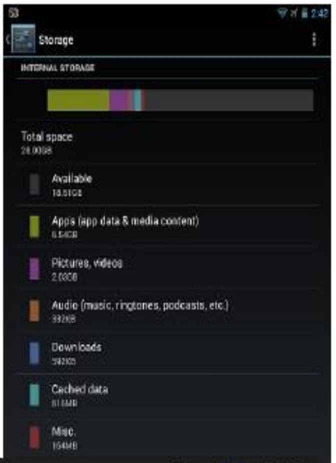
Mahmoud El-Gayyar

elgayyar@ci.suez.edu.eg

Elgayyar.weebly.com

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***


Storage

INTERNAL STORAGE

Total space
28.00GB

Available
18.51GB

Apps (app data & media content)
6.54GB

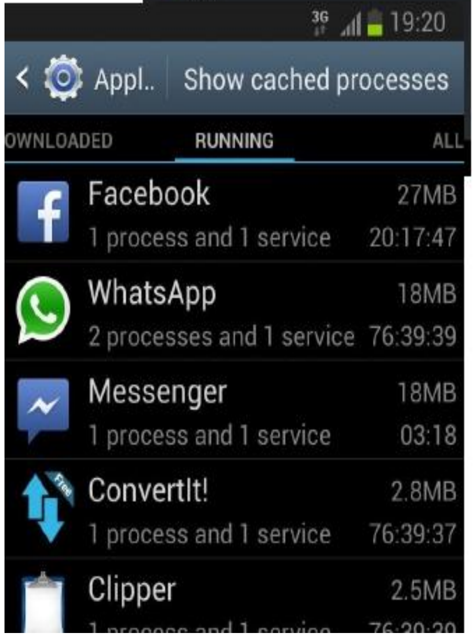
Pictures, videos
2.03GB

Audio (music, ringtones, podcasts, etc.)
332KB

Downloads
592KB

Cached data
61.6MB

Misc.
164MB


19:20

Appl.. Show cached processes

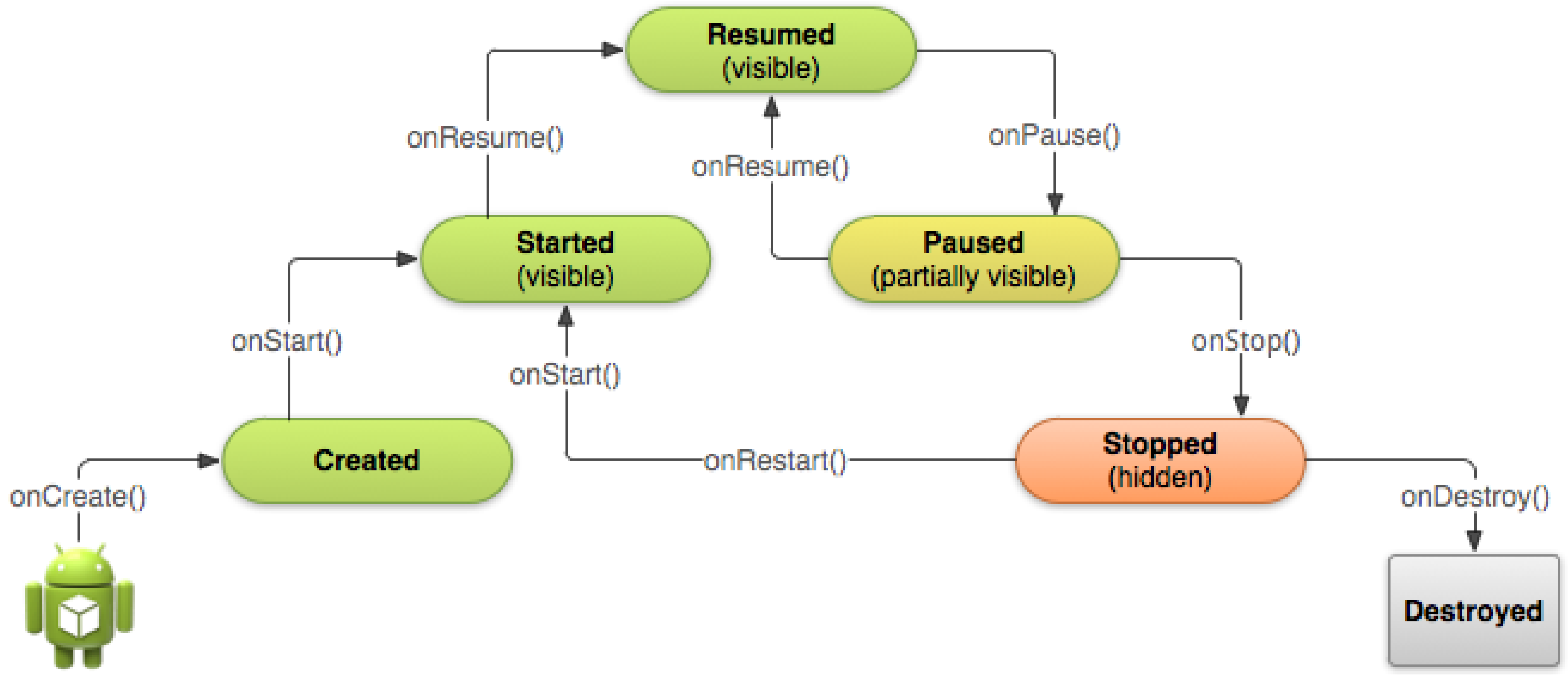
OWNLOADED RUNNING ALL

Facebook	1 process and 1 service	27MB 20:17:47
WhatsApp	2 processes and 1 service	18MB 76:39:39
Messenger	1 process and 1 service	18MB 03:18
ConvertIt!	1 process and 1 service	2.8MB 76:39:37
Clipper	1 process and 1 service	2.5MB 76:39:39

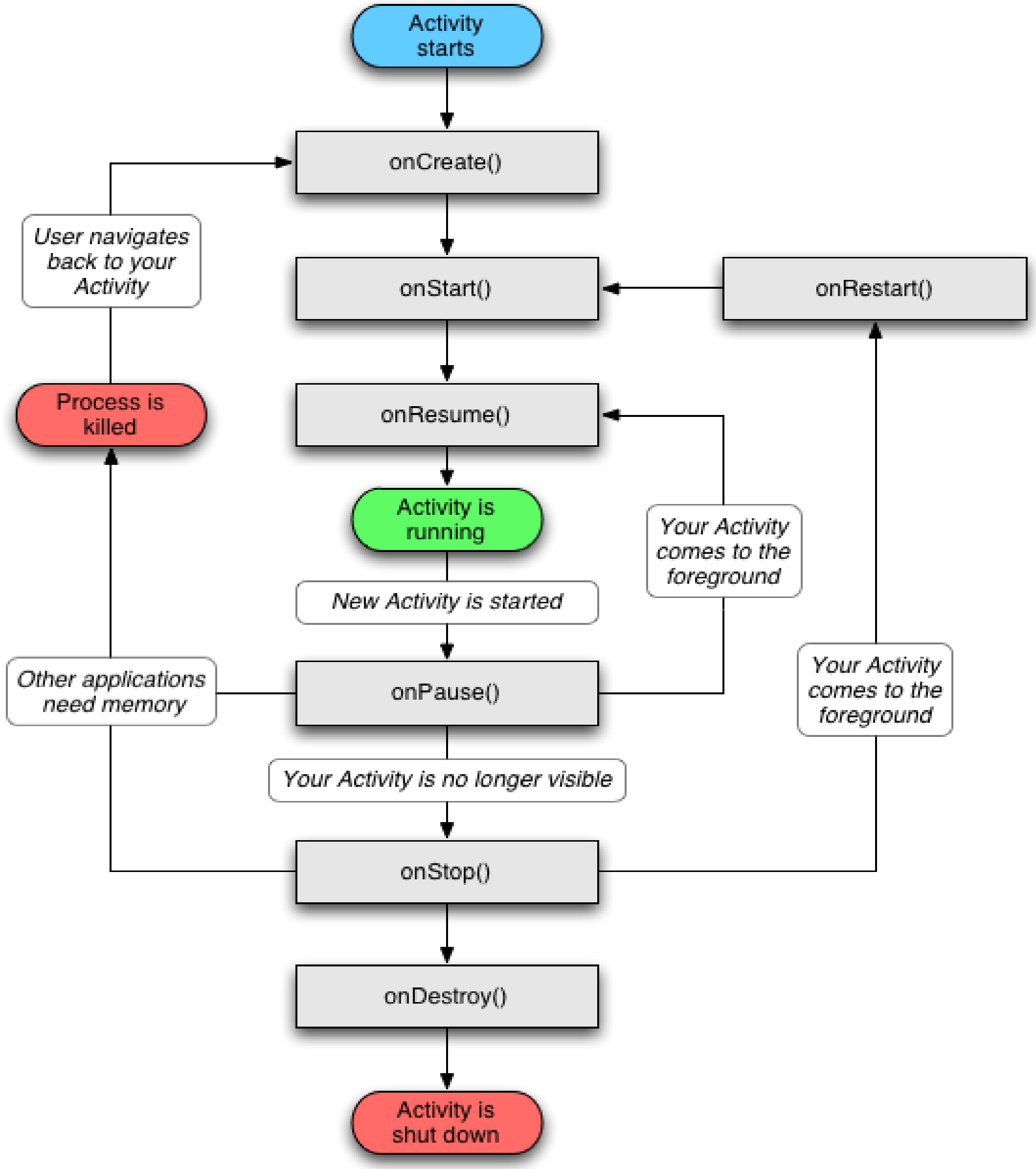
Activity State

- *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  - ***stopped:*** No longer active, but still in the device's active memory.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, ...

Activity Lifecycle 1

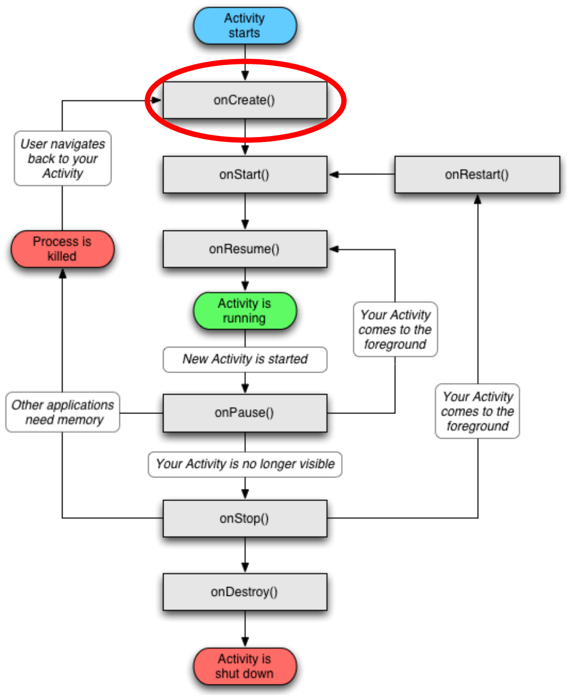


Activity Lifecycle 2


Activity starts
onCreate()
User navigates back to your Activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
Your Activity comes to the foreground
New Activity is started
Other applications need memory
onPause()
Your Activity comes to the foreground
Your Activity is no longer visible
onStop()
onDestroy()
Activity is shut down

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

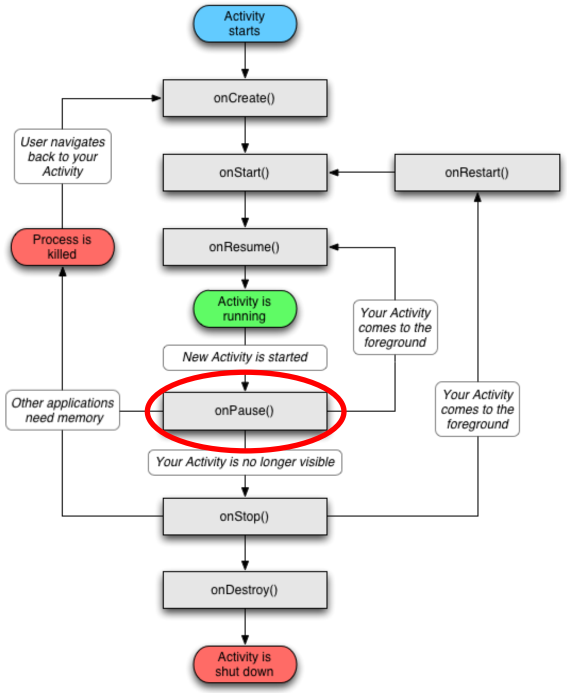


Activity starts
onCreate()
User navigates back to your Activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
Your Activity comes to the foreground
New Activity is started
Other applications need memory
onPause()
Your Activity comes to the foreground
Your Activity is no longer visible
onStop()
onDestroy()
Activity is shut down

```
public class FooActivity extends Activity {
     ...
     public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);          // always call super
          setContentView(R.layout.activity_foo);       // set up layout
          any other initialization code;               // anything else you need
     }
}
```

The onPause Method

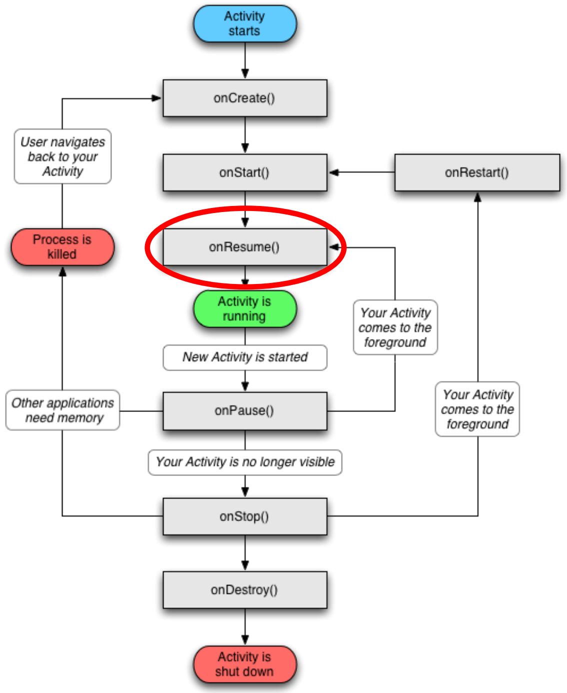
- *When onPause is called, your activity is still <u>partially</u> visible.*
- *May be temporary, or on way to termination.*
  - Stop animations or other actions that consume CPU.
  - Commit unsaved changes (e.g. draft email).
  - Release system resources that affect battery life.


Activity starts
onCreate()
User navigates back to your Activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
Your Activity comes to the foreground
New Activity is started
Other applications need memory
onPause()
Your Activity comes to the foreground
Your Activity is no longer visible
onStop()
onDestroy()
Activity is shut down

```
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.

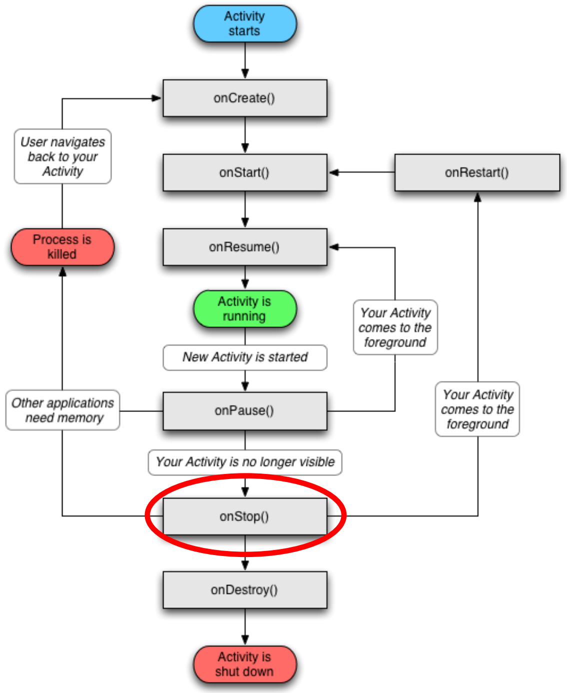

Activity starts
onCreate()
User navigates back to your Activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
Your Activity comes to the foreground
New Activity is started
Other applications need memory
onPause()
Your Activity comes to the foreground
Your Activity is no longer visible
onStop()
onDestroy()
Activity is shut down

```
public void onResume() {
     super.onResume();                        // 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
      ...
}
```

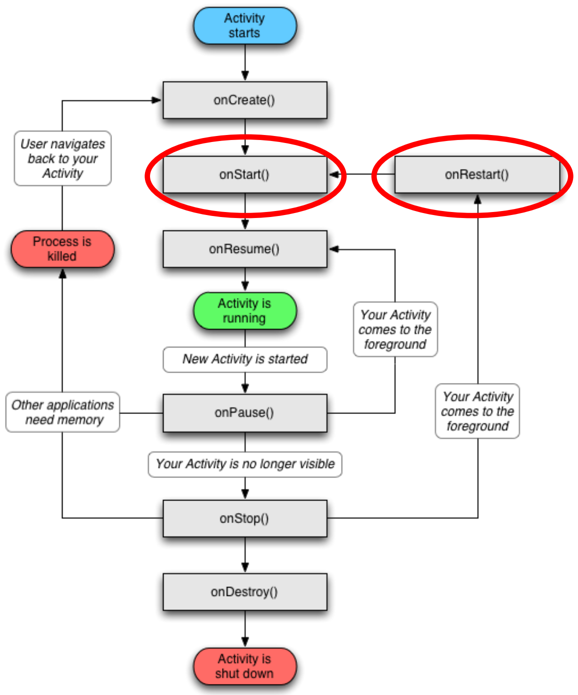


Activity starts
onCreate()
User navigates back to your Activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
Your Activity comes to the foreground
New Activity is started
Other applications need memory
onPause()
Your Activity comes to the foreground
Your Activity is no longer visible
onStop()
onDestroy()
Activity is shut down

The onStart/onRestart Methods

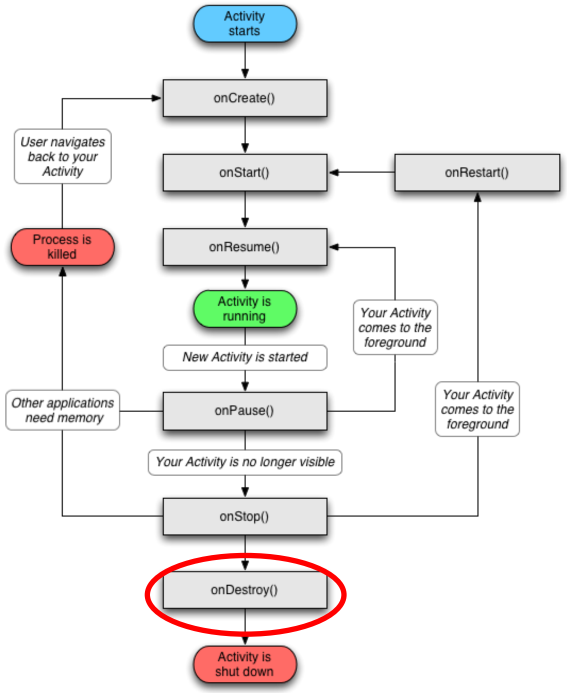
- *onStart is called every time the activity begins*
- *onRestart is called when activity was stopped but is started again later (all but the first start).*
  - Not as commonly used; favor **onResume**.
  - Re-open any resources that onStop closed

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


Activity starts
onCreate()
User navigates back to your Activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
Your Activity comes to the foreground
New Activity is started
Other applications need memory
onPause()
Your Activity comes to the foreground
Your Activity is no longer visible
onStop()
onDestroy()
Activity is shut down

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.


Activity starts
onCreate()
User navigates back to your Activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
Your Activity comes to the foreground
New Activity is started
Other applications need memory
onPause()
Your Activity comes to the foreground
Your Activity is no longer visible
onStop()
onDestroy()
Activity is shut down

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

Testing Activity States

- *Use the LogCat system for logging messages when your app*
  - 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
`Log.d("tag","message");`	Debug message (for debugging)
`Log.e("tag","message");`	Error message(fatal error)
`Log.i("tag","message");`	Info message
`Log.v("tag","message");`	Verbose message(rarely shown)
`Log.w("tag","message");`	Warning message(non-fatal error)
`Log.wtf("tag",exception);`	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);
}
```