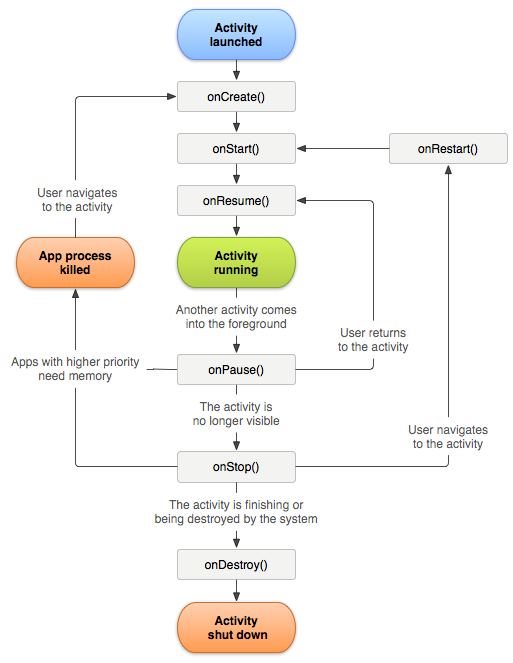
**Teori Activity**

1. Activity merupakan sebuah komponen di Android yang berfungsi untuk menampilkan user*interface* ke layar handset Android pengguna. Ini seperti pada saat Anda melihat daftar percakapan pada aplikasi *chat* atau daftar *email* pada aplikasi Gmail di ponsel Android Anda.
2. Umumnya dalam sebuah aplikasi terdapat lebih dari satu activityyang saling terhubung dengan tugas yang berbeda-beda.
3. Activity merupakan salah satu komponen penting Android yang memiliki daur hidup (life cycle) dalam sebuah *stack* pada *virtual sandbox*yang disiapkan oleh *Dalvik Virtual Machine* (DVM) atau *Android Runtime* (ART) yang bersifat *last in first out*.
4. Pada implementasinya, activity selalu memiliki satu layout *user interface*dalam bentuk berkas xml.
5. Suatu aplikasi Android bisa memiliki lebih dari satu activitydan harus terdaftar di berkas AndroidManifest.xml sebagai sub aplikasi.
6. Sebuah class Java dinyatakan sebuah activityjika mewarisi (*extends*) superclassActivityatau turunannya seperti AppCompatActivitydan FragmentActivity.

Untuk lebih mendalami activity, kami menyarankan Anda untuk membaca referensi berikut :

* [Activities](https://developer.android.com/guide/components/activities.html)

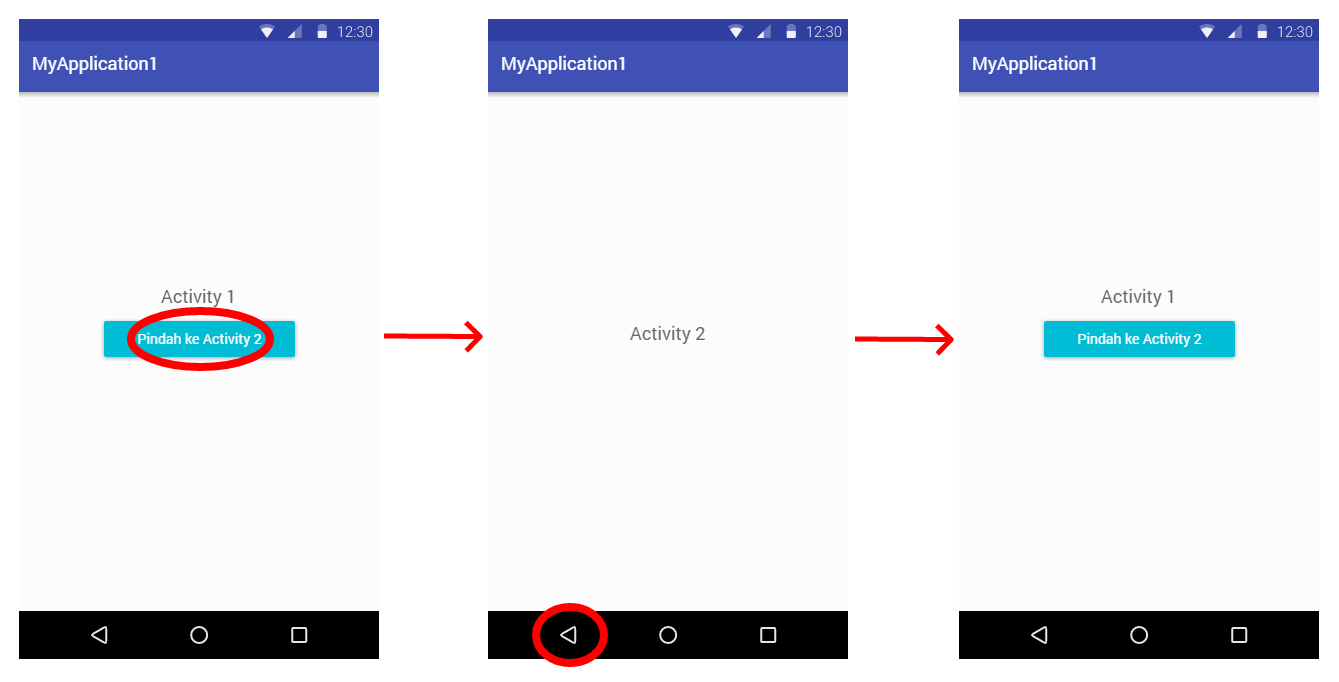
**Activity Lifecycle**

****

Developer yang baik harus mengetahui secara detail tentang *life cycle* sebuah activity. Terutama untuk melakukan aksi yang tepat, saat terjadi perubahan *state* activity. *Callback methods*yang ada dapat digunakan untuk melakukan beragam proses terkait state dari activity.Misalnya melakukan semua inisialisasi komponen di onCreate(), melakukan *disconnect*terhadap koneksi ke *server* pada onStop() atau onDestroy() dan lain sebagainya.

Pemahaman yang baik tentang daur hidup activity akan membuat implementasi rancangan aplikasi Anda menjadi lebih baik. Hal ini juga akan meminimalisir terjadinya *error*/*bug*/*force* *close* yang tidak diinginkan.

**Last In, First Out (LIFO)**



| **Gambar 1** | **Gambar 2** | **Gambar 3** |
| --- | --- | --- |
| Aktif: Activity 1  **onCreate()** → **onStart()** → **onResume()** | Aktif: Activity 2  Stack append: Activity 2 [ **onResume()**] | Activity 1  **onStop()** → **onRestart()** → **onStart()** → **onResume()** |
| Aksi: Klik Button1 (Pindah) | Aksi: Klik Hardware Back Button | Aktif: Activity 1 |
| Stack append: Activity 1 [ **onStop()**] | Activity 2 [ **finish()**]  Stack pop: Activity 2 [ **onDestroy()**] |  |

1. **Gambar 1**  
   Jika Anda memiliki sebuah aplikasi yang terdiri dari 2 activity, maka activity pertama akan dijalankan setelah pengguna meluncurkan aplikasi melalui ikon aplikasi di layar device. Activity yang ada saat ini berada pada posisi activity*running* setelah melalui beberapa *state* onCreate (*created*) → onStart (*started*) → onResume (resumed) dan masuk ke dalam sebuah stack activity.  
   Bila pada activity pertama Anda menekan sebuah tombol untuk menjalankan activity kedua, maka posisi *state* dari activity pertama berada pada posisi *stop*. Saat itu, *callback* onStop() pada activity pertama akan dipanggil.  
   Ini terjadi karena activity pertama sudah tidak berada pada layar *foreground* / tidak lagi ditampilkan. Semua informasi terakhir pada activity pertama akan disimpan secara otomatis.  
   Sementara itu**, activity** kedua masuk ke dalam *stack* dan menjadi activity terakhir yang masuk.
2. **Gambar 2**  
   Activity kedua sudah muncul di layar sekarang. Ketika Anda menekan tombol *back* pada *physical button* menu utama atau menjalankan metode finish(), makaactivity kedua Anda akan dikeluarkan dari *stack*.  
   Pada kondisi di atas, state activity kedua akan berada pada *destroy*. Oleh karenanya, metode onDestroy() akan dipanggil.  
   Kejadian keluar dan masuk *stack* pada proses di atas menandakan sebuah model *Last In, First Out*.Activity kedua menjadi yang terakhir masuk *stack* (*Last In*) dan yang paling pertama keluar dari *stack* (*First Out*).
3. **Gambar 3**Activity pertama akan dimunculkan kembali di layar setelah melalui beberapa *state* dengan rangkaian *callback method* yang terpanggil, onStop → onRestart → onStart → onResume.

Detailnya dapat Anda baca di sini :

* [LifeCycle](https://developer.android.com/guide/components/activities.html#Lifecycle)

**Saving Activity State**

Ketika sebuah activity mengalami *pause* kemudian *resume*, maka state dari sebuah activity tersebut dapat terjaga. Sebabnya, obyek activity masih tersimpan di *memory* sehingga dapat dikembalikan *state*-nya.

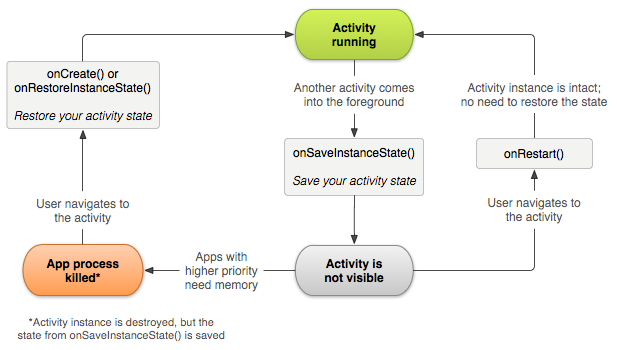
Dengan menjaga *state* dari activity, maka ketika activitytersebut ditampilkan, kondisinya akan tetap sama dengan kondisi sebelumnya.

Akan tetapi ketika sistem menghancurkan activityuntuk keperluan memori misalnya karena memori habis, maka obyek activity dihancurkan. Alhasil, ketika activityingin ditampilkan kembali diperlukan proses *recreate* activity yang dihancurkan tadi.

Kejadian di atas adalah hal yang lumrah terjadi. Oleh karena itu, perubahan yang terjadi pada activity perlu disimpan terlebih dahulu sebelum ia dihancurkan. Di sinilah metode onSaveInstanceState() digunakan.

Dalam onSaveInstanceState terdapat bundle yang dapat digunakan untuk menyimpan informasi. Informasi dapat disimpan dengan memanfaatkan fungsi seperti putString() dan putInt().

Ketika activity di-*restart*,bundle akan diberikan kepada metode onCreate dan onRestoreInstanceState. Bundle tersebut akan dimanfaatkan untuk mengembalikan kembali perubahan yang telah terjadi sebelumnya.



Proses penghancuran activitydapat juga terjadi ketika terdapat perubahan konfigurasi seperti perubahan orientasi layar (*portrait-landscape*), *keyboard availability*, dan perubahan bahasa. Penghancuran ini akan menjalankan *callback method* onDestroy dan kemudian menjalankan onCreate. Penghancuran ini dimaksudkan agar activitydapat menyesuaikan diri dengan konfigurasi baru yang muncul pada kejadian-kejadian sebelumnya.

Hal yang perlu diingat ketika menggunakan onSaveInstanceState adalah untuk tidak menyimpan data yang besar pada bundle. Contohnya, hindari penyimpanan data *bitmap* pada bundle. Bila data pada bundleberukuran besar, proses serialisasi dan deserialisasi akan memakan banyak memori.