# General lines about Android Programming

## Android SDK

The Android software development kit (SDK) includes a comprehensive set of development tools.
These include a debugger, libraries, a handset emulator based on QEMU, documentation, sample code, and tutorials.
Enhancements to Android's SDK go hand in hand with the overall Android platform development.
Each Android Version has a specific set of API, named "API level".
Android applications are packaged in .apk format and stored int the /data/app folder on the Android OS.
APK package contains .dex files ( executable for the dvm ), resource files, etc.

## Android NDK

The NDK is a toolset that allows you to implement parts of your app using native-code languages ( C/C++ ).
This can be helpful in reusing existing code libraries written in these languages.
Most apps do not need the Android NDK.

## Android Debug Bridge

The Android Debug Bridge (ADB) is a toolkit included in the Android SDK package.
It consists of both client and server-side programs that communicate each other.
The ADB is typically accessed through the command-line interface.
A typical use is to compile a source file and run it on a physical device, without the use of the android emulator.

## Project Sections

The project is composed of three main directory :

- **Gen**
  This folder contains Java Library automatically generated.
  In particular, an "R.java" file, containing exadecimal values referreing to gui elements.
- **Src**
  This directory contatins the source code of the application.
  In Java it corresponds to the default source package.
- **Res**
  This directory contains all the resources used by the application.
  Its subdirectory, "drawable", contains all the graphics resource such as image and so on.
  The GUI Layout is defined in a xml file, contained in this directory.

## Activity

### The concept

An activity is a single, focused thing that the user can do.
Almost all activities interact with the user, so the Activity class takes care of managing graphics resources.
Every application is build around a single instance of the Activity class.
All the activity classes must have a corresponding tag in their package's manifest file.

### Some Methods

- **onCreate(), onStart(), onResume(), onDestroy(), ...**
  As easy to understand, these are methods used to interact with the flow of the application lifecycle.
  Override these methods to initialize it on start or commmit modifications when exiting.

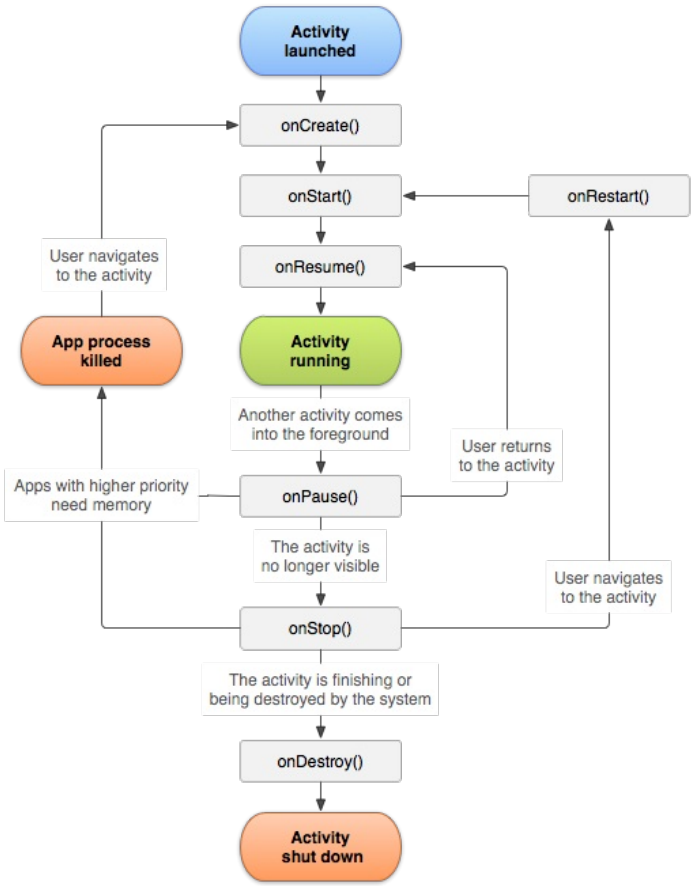
- **setContentView()** Set the activity content to an explicit view.
  This view is placed directly into the activity's view hierarchy. It can itself be a complex view hierarchy.
  Parameters :
  - view: the desired content to display.
  - params: layout parameters for the view.

- **onActivityResult()** Let's suppose an application needs the user to select a contact from the contect list.
  Calling another application is as simple as this :

```
startActivityForResult(
    new Intent( Intent.ACTION_PICK,
        new Uri( "content://contacts" )
    ),
    PICK_CONTACT_REQUEST
);
```

  Override onAcitvityResult() let you use the result of the called application, by using an Intent object.

### Lifecycle Diagram



## Manifest

In software packaging, it is common to list the contents of a distribution in a manifest file.
In java this kind of file is used to specify information used when packing in a jar file :

- store hashes of stored files for signature validation.
  This aspect will be deepened in the next charapter, "Packing Applications".
- sealing jar files (i.e. ensure that only classes from this jar file are loaded in the packages defined in this jar file).
- store version/product/producer information to be readable at runtime

This file also contains information about the permission of the application.
The ability to start a particular Activity can be enforced when it is declared in its manifest's <activity> tag.
Other applications will need to declare a corresponding <uses-permission> element in their own manifest to be able to start that activity.

## Layout

A layout defines the visual structure for a user interface, such as the UI for an activity or app widget. You can declare a layout in two ways:

- Declare UI elements in XML.
  Android provides a straightforward XML vocabulary that corresponds to the View classes, such as those for widgets and layouts.
- Instantiate layout elements at runtime.
  Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

The Android framework allows you to use either or both of these methods for declaring and managing your application's UI.
For example, you could declare your application's default layouts, screen elements, and their properties in a XML file,
and then modify the state of those object in yout code, at run time.