University of Mila
Computer Science Department

Third-year Bachelor
Mobile Applications Course

Chapter 3

Dr. Hamza MEGUEHOUT

2025/2026

A **mobile application** differs from **desktop applications**

For example, the user's interaction with the application

does not always start in the same place.

The user journey often begins in a **non-deterministic** way

If you open a messaging application from your home screen,

you may see a **list of emails**.

However, if you use another application that then launches your messaging application,

you may be taken directly to the screen for composing an email

When one application calls another,

the calling application invokes an **ACTIVITY** in the other application,

rather than **the entire application itself**

Activity Concept

A fundamental component of Android

Its main role is to interact with the user

For example, to have two screens (a selection screen and a services display screen), you will usually have two activities:

1. The **first** one manages the **selection** part;
2. The **second** one manages the **display** of services.

Most applications contain **multiple** screens

Each **screen** in an application represents an **activity**

They consist of **several activities**

(only one activity is launched at a time)

In general, an activity in an application is defined as the **main activity**, which is the **first screen** displayed when the user launches the application.

Each activity can then start **another activity** in order to perform different actions.

Messaging application

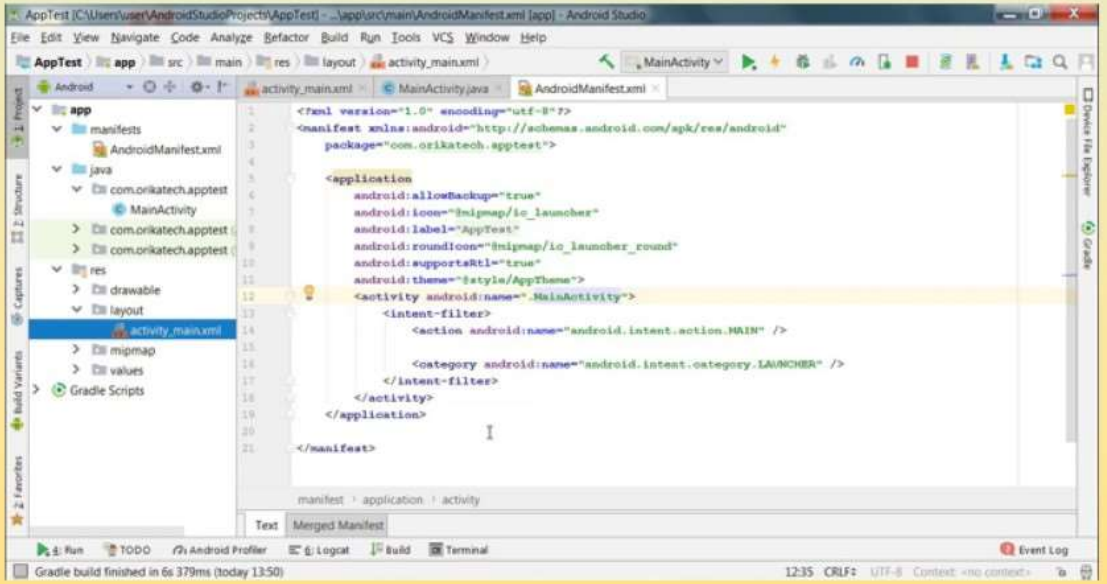
Main activity

Email inbox

The main activity can launch other activities that provide different screens such as **composing emails, opening individual emails**, etc.

To use activities in your application, you must

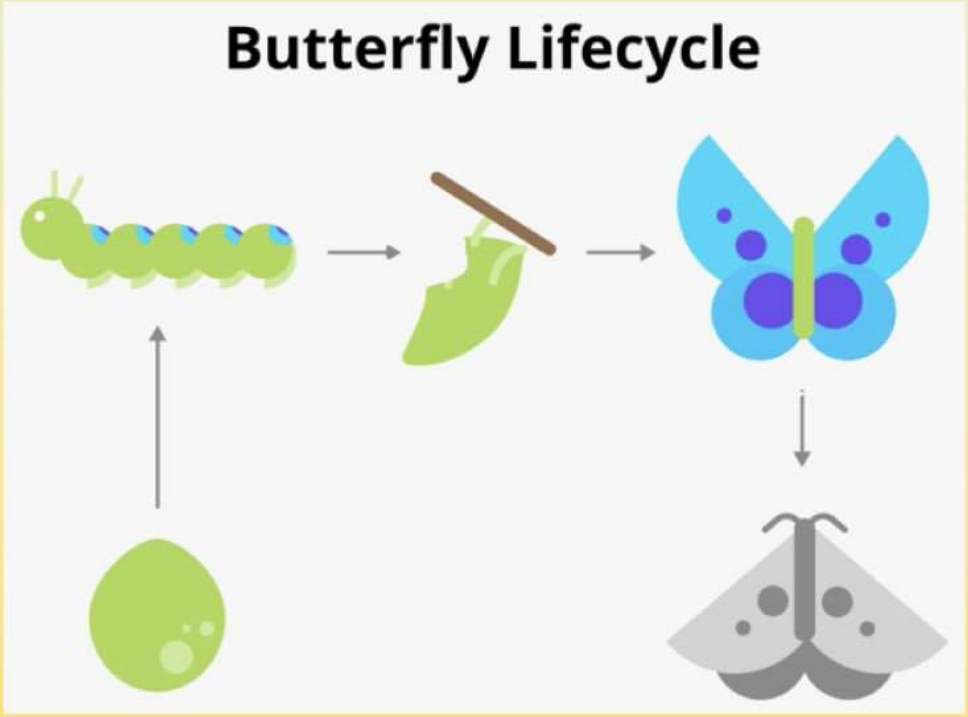
- Declare the information in the application **manifest**
- Properly manage the **activity lifecycle**



Activity Lifecycle

The user has the impression that the screen displayed remains the same as long as the application is running

However, in reality, the associated activity may be **destroyed** and **recreated** several times during the application's lifecycle


Butterfly Lifecycle

Why???

The Android environment is generally more limited in resources than a traditional computer.

Therefore, it is important to **minimize the use of these resources** (battery, memory, etc.).

To achieve this, the Android system manages an activity lifecycle that prioritizes resource efficiency.

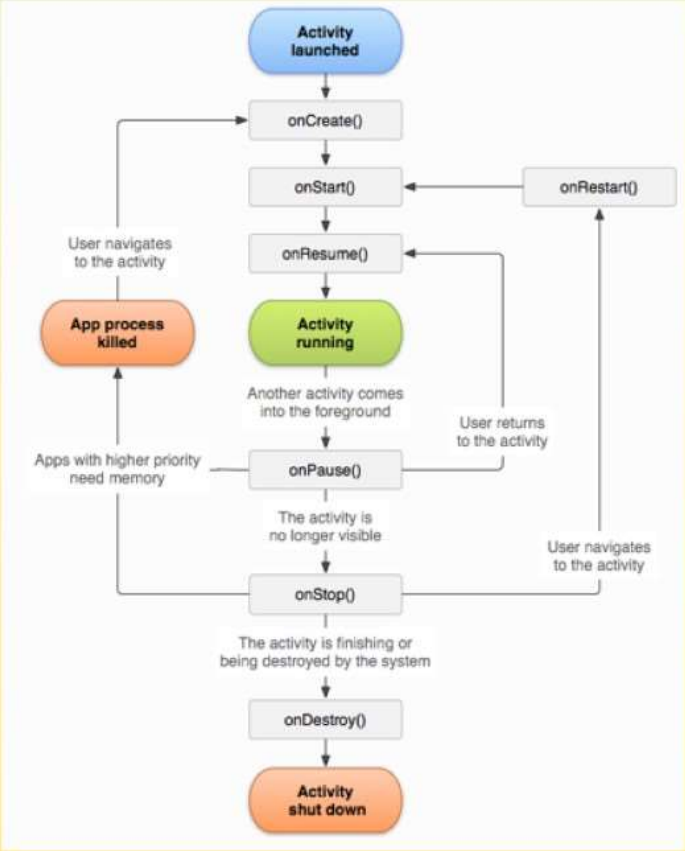
When the user **answers a phone** call while an **activity is running**

it is **suspended** by the system. If necessary, the system may decide to **terminate** it to save resources.

When the user **returns** to the application, the activity will be completely **recreated**.

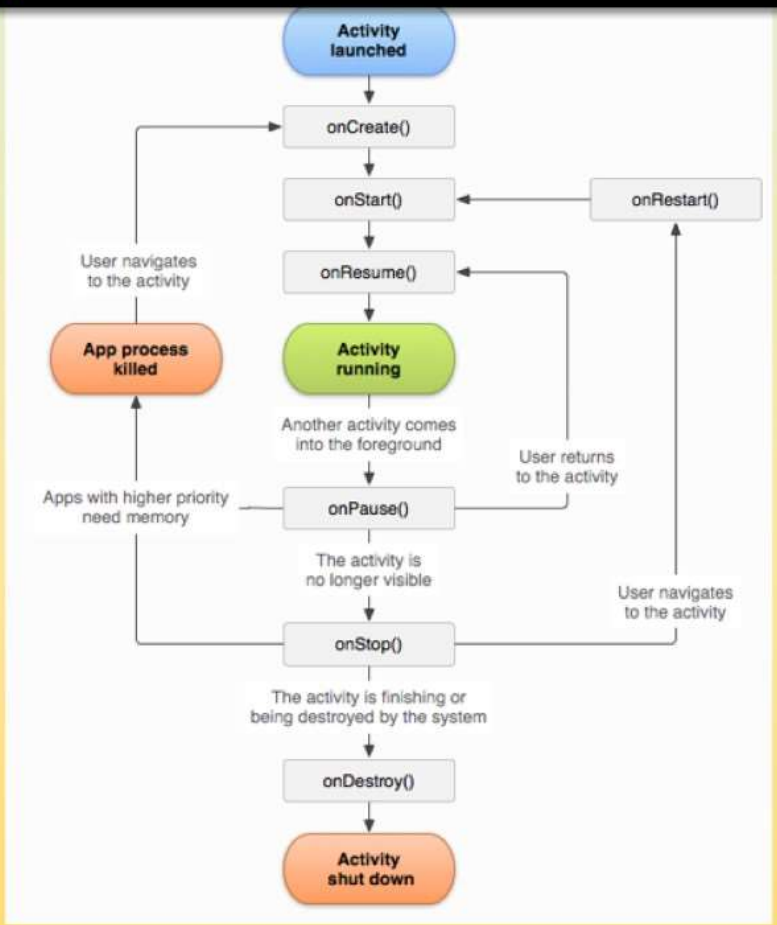
The set of states that an activity can go through during its lifetime, from its **creation** to its **destruction** and the **release** of its resources by the system.

When a user navigates between the activities of your application (as well as outside of it), the activities **move** from one state to another during their lifecycle.



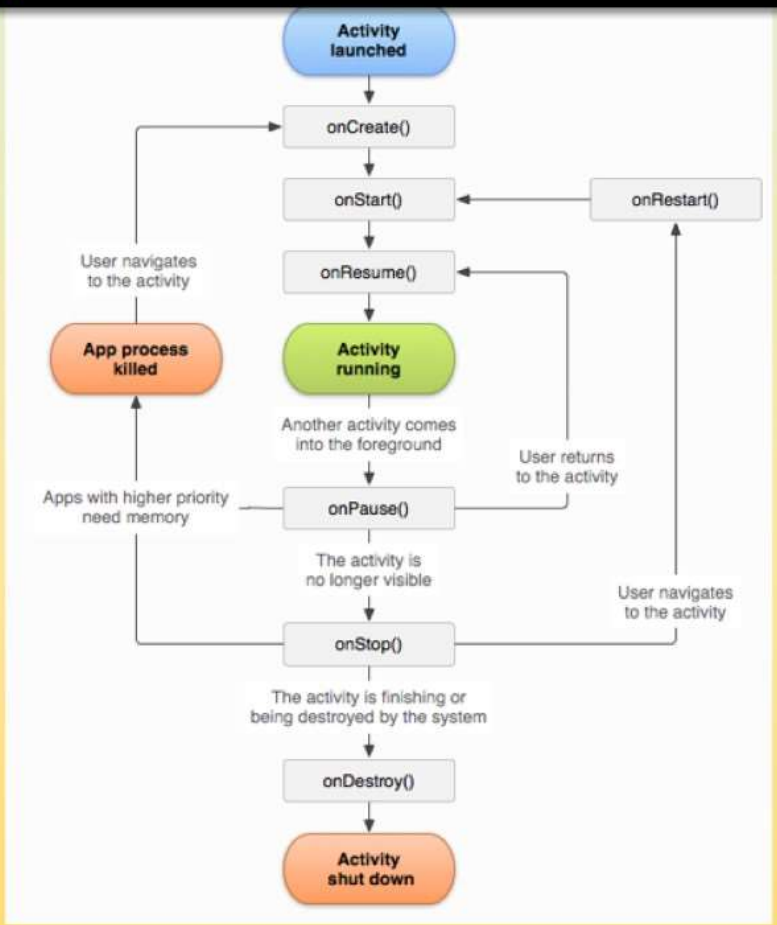
Email viewing application → **Activity** to display the list of emails

The activity is **created** by the system, and its methods **onCreate**, **onStart**, and **onResume** are called in sequence.The activity is then displayed on the screen.



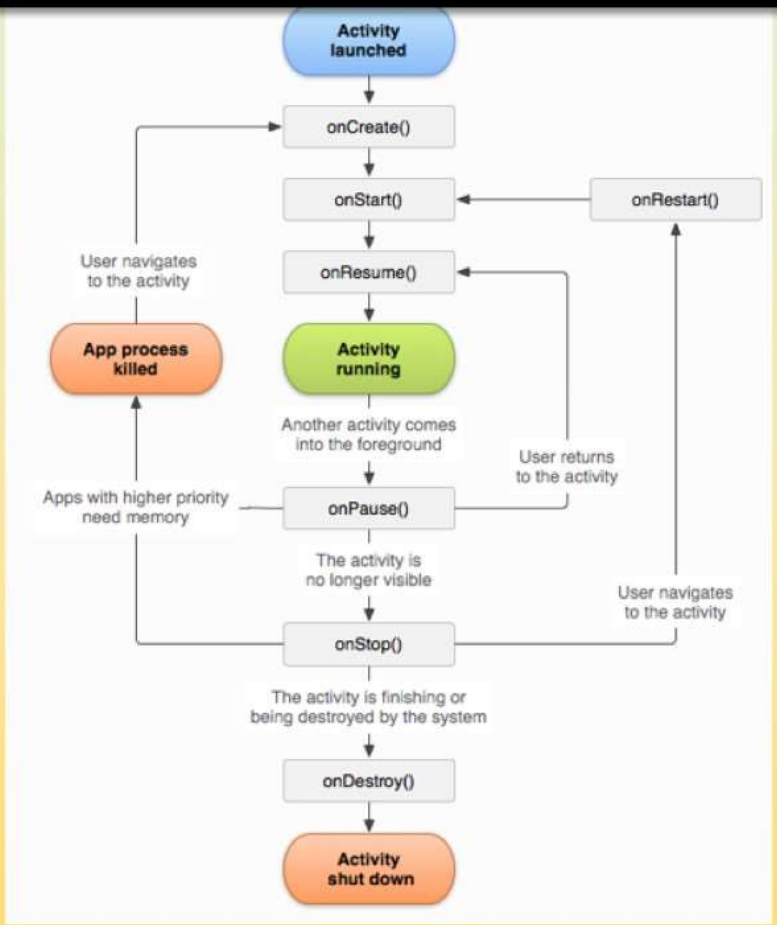
The user clicks on an email to view its details → This action **creates** and **displays** a **new activity**.

The **email list activity** is **stopped** (**onPause** then **onStop** are called) → it is no longer in the foreground.



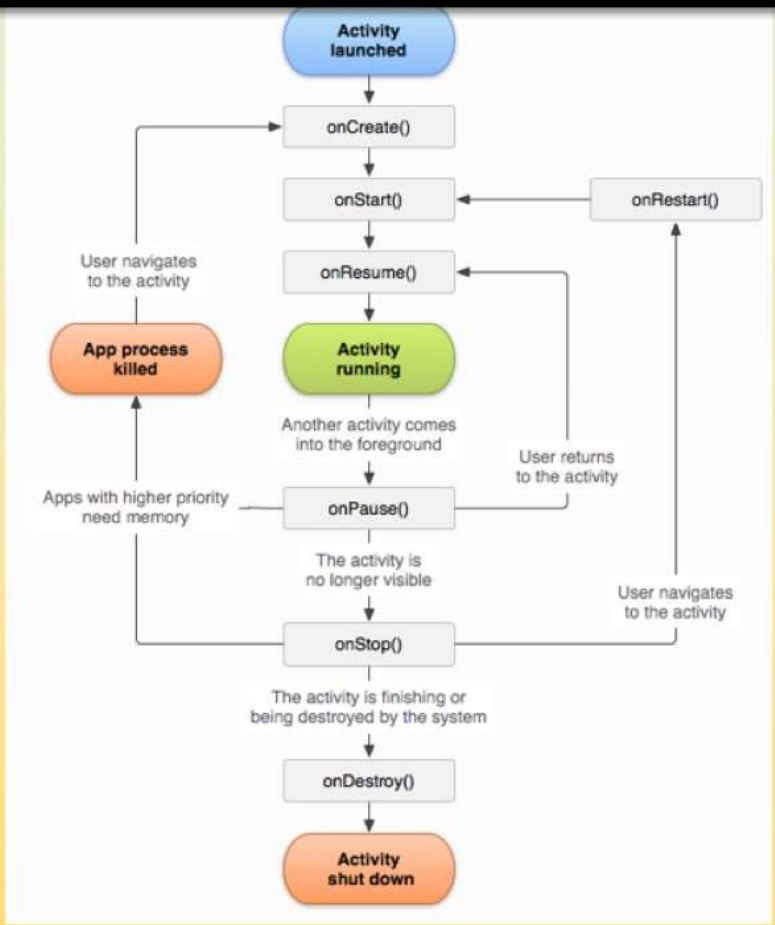
The user presses the **Back button** to return to the email list.

The activity must be **displayed again**, and its methods **onRestart**, **onStart**, and **onResume** are called before that.



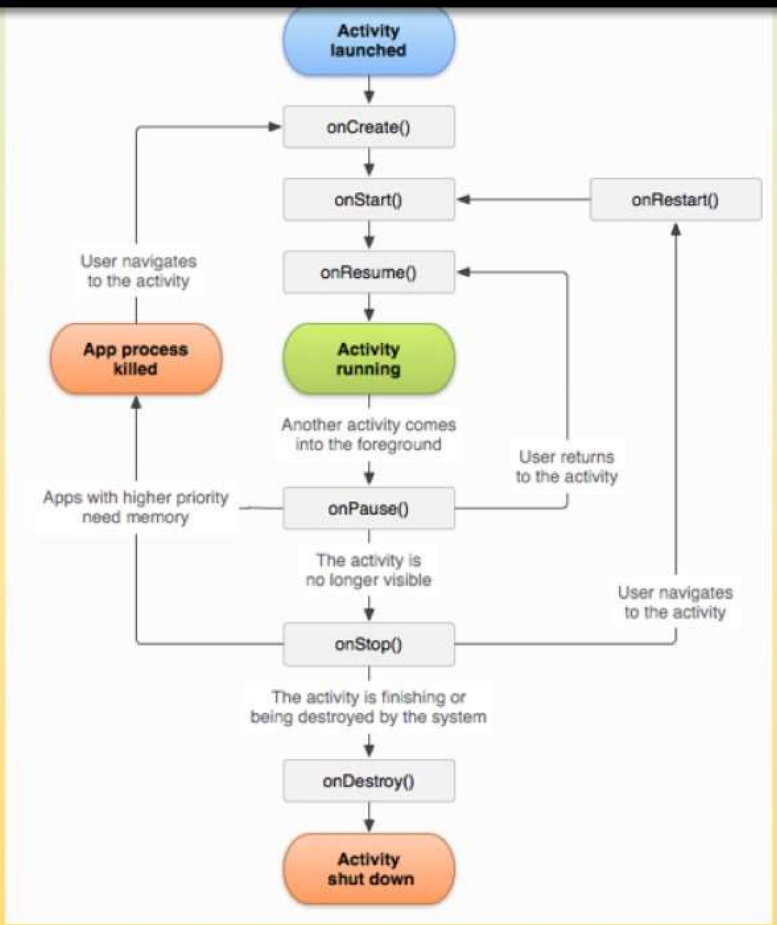
The user **receives** a phone call and answers it → the phone application comes to the foreground.

The email list activity is **stopped again** (**onPause**, **onStop** are called).



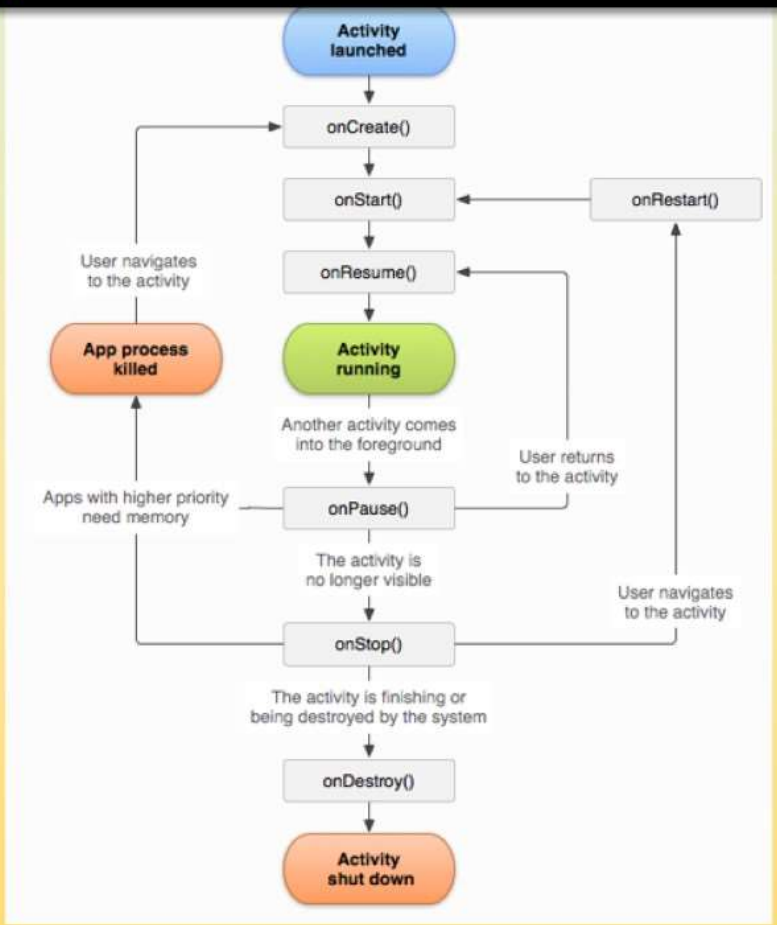
The phone call lasts for some time → the system decides to **free up resources**.

When the user hangs up and returns to the email application, the activity is completely **recreated** (**onCreate**, **onStart**, **onResume** are called).



The user decides to **close** the email application.

The activity is then completely **destroyed** (**onDestroy** is called).



onCreat

It is called when your activity is created by the system. Generally, the operations performed in this method are used to set up **the user interface**, **initialize variables**, etc.

It is in this method that you should perform tasks that need to be **executed only once during the entire lifecycle of the activity**.

At this stage, the activity is created, **but the user cannot yet see it** or interact with its elements.

onStart

It allows you to know when an activity is about to become visible to the user. The **user interface becomes visible**, but **the user still cannot interact with its elements**.

For example, if your activity plays an animation, this method is a good place to start the animation automatically.

onResume

The activity becomes **fully operational**. The user can interact with the application and click on its UI elements. This is the right moment to start resource-intensive operations (such as activating the camera, opening network connections, etc.).

The application **remains in this state as long as there is no interruption**, such as receiving a phone call, starting a new activity, or displaying a dialog box.

onPause

An activity enters this state, for example, when **an AlertDialog is displayed**: the underlying activity is still visible, but it is no longer in the foreground.

This method is the **opposite of onResume()**: everything that was started in onResume() should be paused here.

This **is the right moment to suspend resource-intensive operations** (such as disabling the camera, closing network connections, etc.). Be careful: a **paused activity may still be visible to the user**, so you should not use this method to modify the user interface or stop animations.

This activity can still be destroyed by the system, for example, if the system runs low on memory.

onStop

When a new activity is started, the current activity enters this state. It is therefore **no longer visible to the user**.

If your activity needs to **save information** (for example to disk) in order to restart properly, you can perform these operations at this stage. You can also use this method to **stop graphical operations** (such as animations, video playback, etc.).

onDestroy

It is called when **the activity is destroyed** (either naturally or by the system).

Note: When an activity is in the states **onPause**, **onStop**, or **onDestroy**, it is in a situation where **it may be destroyed to temporarily free system resources**. This does not necessarily mean that the activity has been simply closed by the user. Sometimes, due to system pressure, the activity may be destroyed without even calling **onDestroy()**.

The **onPause()** state is the only one that is guaranteed to execute completely before the activity is fully destroyed. Therefore, this **is the appropriate place to save the application state**.

onRestart

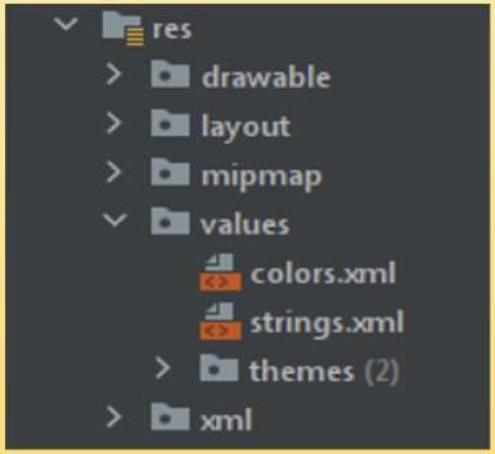
Permets de savoir qu'une **activité va être à nouveau démarrée après avoir été stoppée**.

It allows you to know that an **activity is about to be restarted after having been stopped**.

The call to this method occurs between onStop and onStart. This makes it easier to distinguish **between a normal start of the activity and its restart**.

Resources

We can distinguish several types of resources (images, video, audio, text strings, etc.), and all these resources are located in your application under the **res/** directory.



res\drawable : this directory contains resources that can be drawn. These resources include images (JPEG, GIF, etc.), as well as generic shapes (circles, squares, etc.) defined in XML files, etc.

res\layout : contains the different views required for the user interface layout.

res\values : contains XML files that store simple values such as strings, integers, colors, styles, etc.

Centralize text for reuse.

- Stored in ***res/values/strings.xml***
- Used with ***@string/nom***
- Best practice : aucun texte "en dur" dans les layouts

```
<string name="app_name">MonApp</string>
```

Managing images and XML shapes

- Stored in ***res/drawable/***
- Types : PNG, JPG, XML, etc.
- Example : @drawable/logo

rounded_button.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <solid android:color="@color/second_color" />
    <corners android:radius="30dp" />
</shape>
```

Standardize the appearance of views (color, size, corners, etc.)

- Defined in ***res/values/themes.xml***
- Used with style="***@style/MonStyle"Parent ....***"
- Advantage: easy to apply to multiple components.

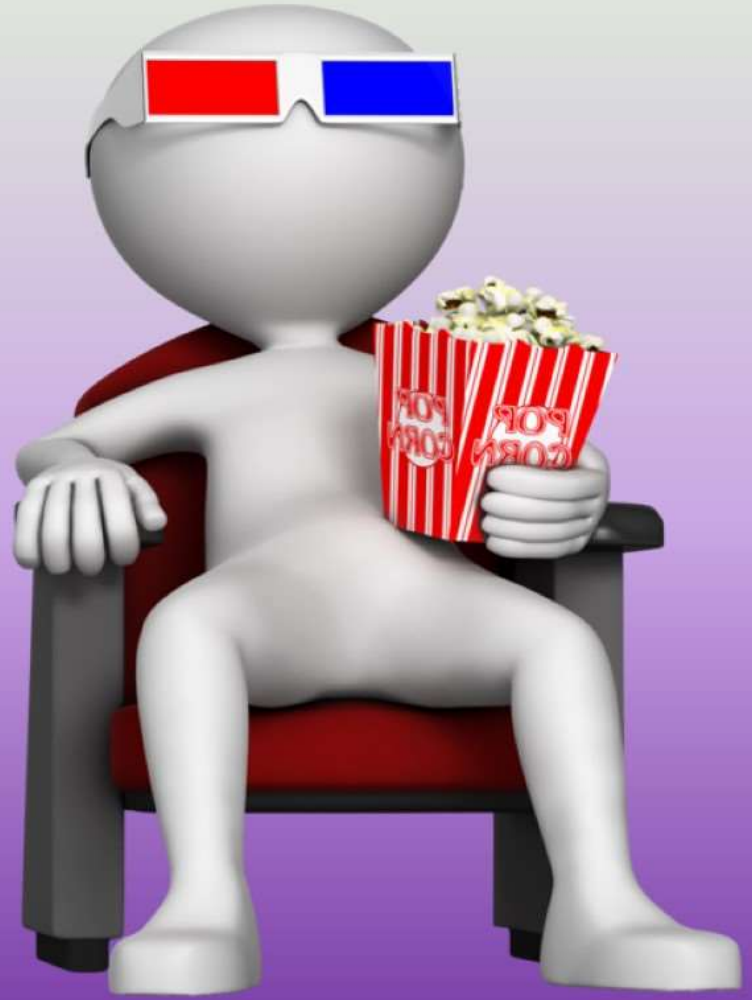
```
<style name="MyCustomButton" parent="TextAppearance.Compat.Notification">
    <item name="android:background">@drawable/rounded_button</item>
    <item name="android:textColor">#FFFFFF</item>
    <item name="android:textSize">18sp</item>
    <item name="android:padding">12dp</item>
    <item name="android:textStyle">bold</item>
</style>
```

```
<style name="Theme.TP_05" parent="android:Theme.Material.Light.NoActionBar" >
    <item name="android:colorBackground">@color/background</item>
    <item name="android:buttonStyle">@style/MyCustomButton</item>
</style>
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Enter"
    style="@style/MyCustomButton"
    />
```

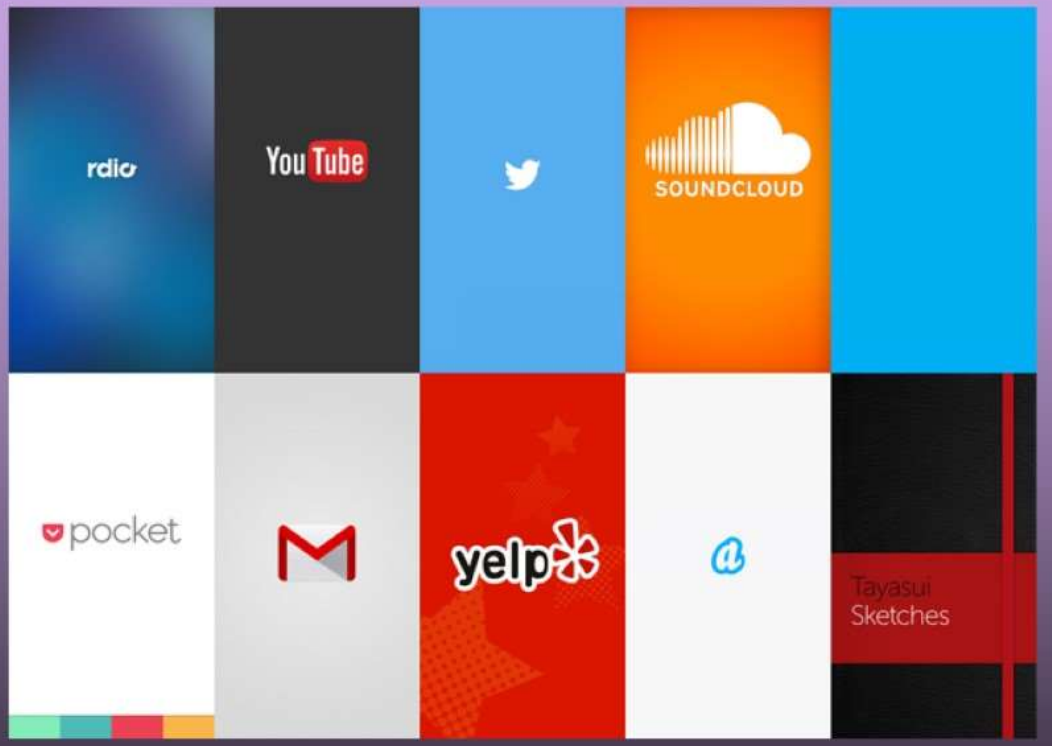
Add motion to views

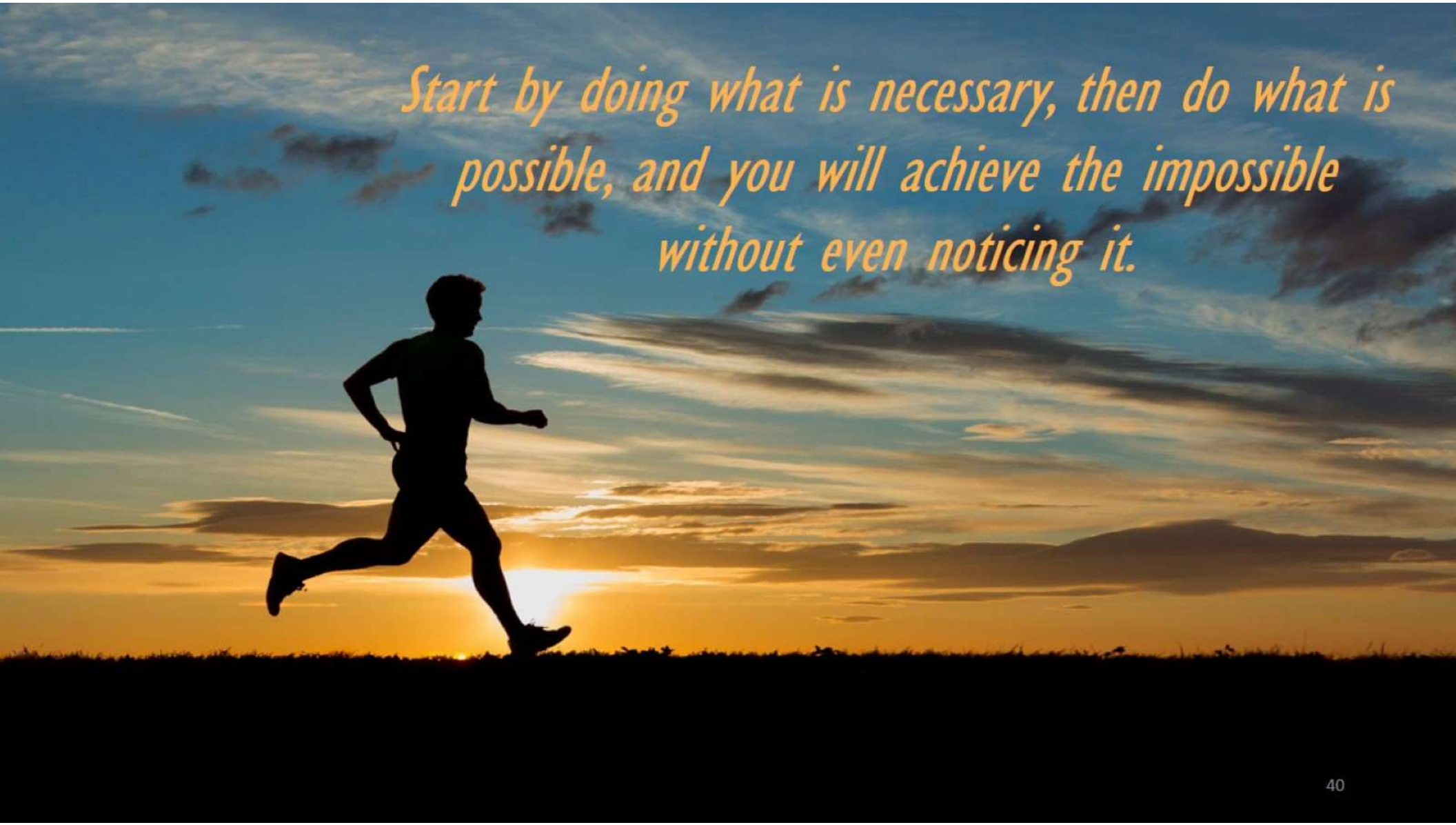
- Types: alpha, scale, rotate, translate
- Defined in res/anim/
- Used with AnimationUtils.loadAnimation()



Did you know?

Splash




Start by doing what is necessary, then do what is possible, and you will achieve the impossible without even noticing it.