

République Algérienne Démocratique et Populaire  
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

**Abdelhafid Boussouf University Center of Mila**

**3<sup>ème</sup> année Licence**

**Cours Applications Mobiles**



# Chapitre 03 et 04

**Responsable Matière**

**Dr. MEGUEHOUT Hamza**



L'application **MOBILE** diffère des applications **BUREAU**

Ex : l'interaction d'un utilisateur avec l'application !

**ne commence pas toujours au même endroit ...**

Le parcours de l'utilisateur commence souvent de manière  
**non déterministe**







Si vous ouvrez une application de messagerie à partir de votre écran d'accueil  
**vous pouvez voir une liste d'e-mails**

Si vous utilisez une autre application qui lance ensuite votre application de messagerie  
**vous pouvez accéder directement à l'écran de l'application de messagerie pour  
composer un e-mail**





Lorsqu'une application en appelle une autre  
l'application appelante invoque une **ACTIVITÉ** dans l'autre application  
plutôt que l'application dans son ensemble atomique





# Notion d'Activité





# Composante fondamentale d'Android

## Son rôle principal d'interagir avec l'utilisateur

Pour avoir deux écrans (un écran de **sélection** et un écran d'**affichage** des services).  
Vous aurez généralement deux activités :

1. La **première** qui gère la partie **sélection** ;
2. La **seconde** qui gère **l'affichage des services**.







La plupart des applications contiennent plusieurs écrans



Chaque écran de l'application représente une activité



Elles comprennent plusieurs activités  
(Une seule activité est lancée à la fois)





En règle générale, une activité dans une application est spécifiée comme **activité principale**, qui est le **premier écran à apparaître** lorsque l'utilisateur lance l'application.

**Chaque activité** peut alors démarrer **une autre activité** afin d'effectuer des actions différentes.







Application de messagerie



L'activité principale



Boîte de réception de courrier électronique



Activité principale peut lancer d'autres activités qui fournissent des écrans  
*la rédaction d'e-mails, l'ouverture d'e-mails individuels, etc.*





Pour utiliser des activités dans votre application :

- Enregistrer des informations dans le **manifeste** de l'application
- Gérer les **cycles de vie** des activités de manière appropriée





# Cycle de vie d'une activité







L'utilisateur a l'impression que l'écran qui lui est présenté existe  
aussi longtemps que l'application s'exécute

Mais dans la réalité, l'activité associée est peut être  
**détruite** et **recréée** plusieurs fois durant la vie de l'application





# Pourquoi ???





L'environnement Android est généralement plus limité en ressources qu'un ordinateur classique

Il est donc important de **limiter** au mieux **l'usage de ces ressources** (*batterie, mémoire, etc.*).

Pour cela, le système **Android** gère un **cycle de vie des activités** en **privilégiant l'économie de ressources**.





L'utilisateur **répond au téléphone** alors qu'une **activité s'exécute**

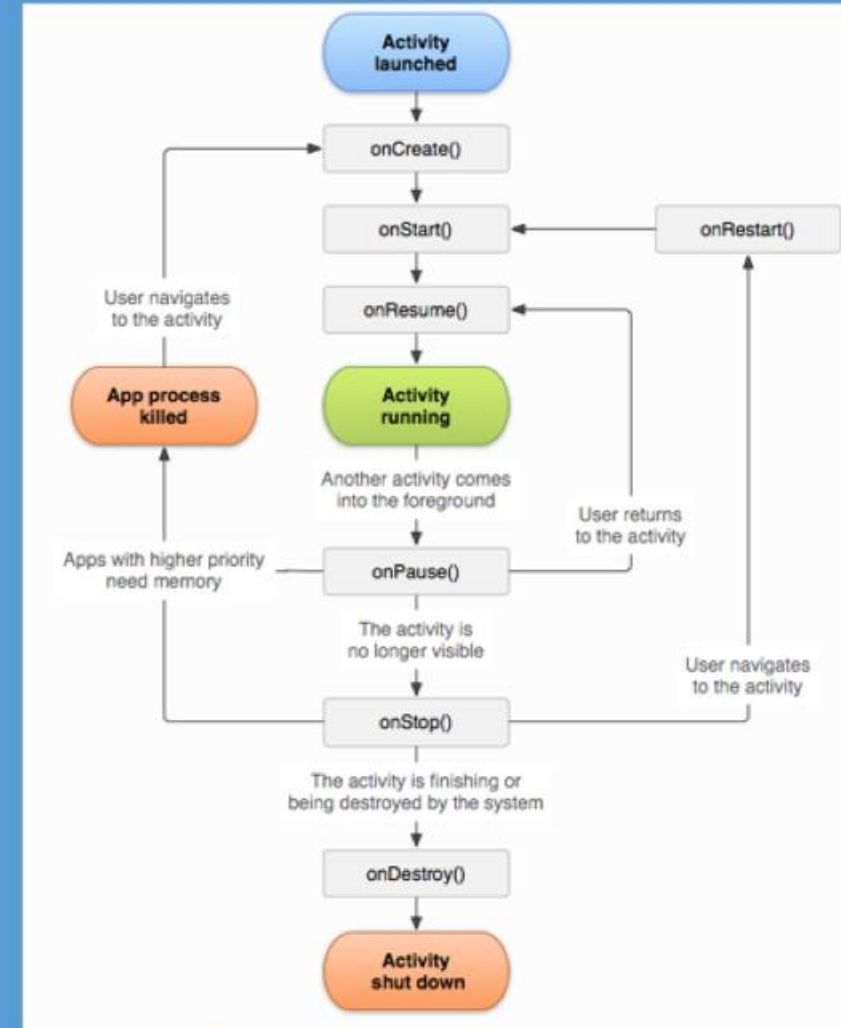


Elle est **SUSPENDUE** par le système et, si nécessaire, le système peut décider de la **SUPPRIMER** pour économiser des ressources



Lorsque l'utilisateur voudra **reprendre** l'utilisation de l'application, l'activité sera intégralement **RECRÉÉE**

Lorsqu'un utilisateur navigue entre les activités de votre application (ainsi que hors de celle-ci), les activités **PASSENT** d'un état à un autre au cours de leur cycle de vie.



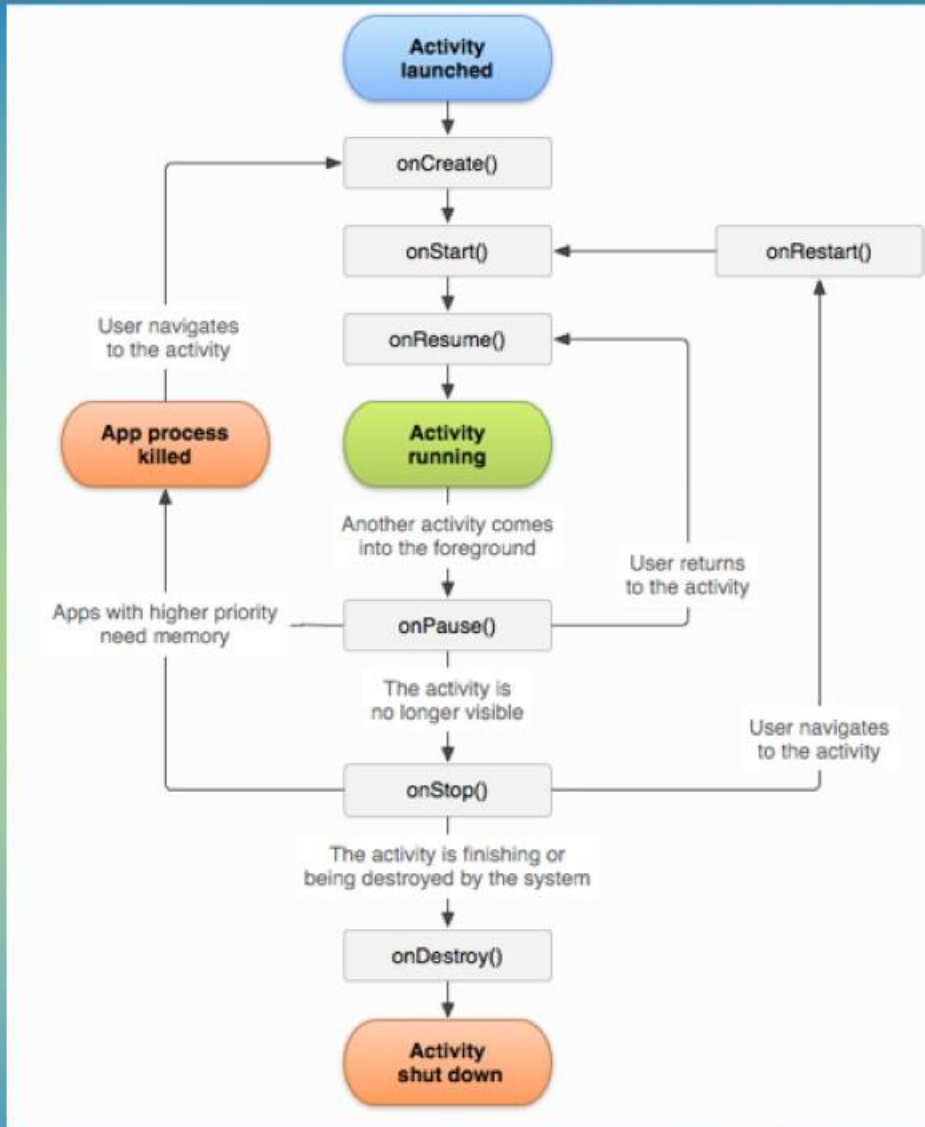


Application de consultation de mail → **activité** pour présenter la liste des mails.

Activité est **CRÉÉE** par le système et ses méthodes **onCreate**, **onStart** et **onResume** sont appelées à la suite.

Activité est affichée à l'écran.

L'utilisateur clique sur un mail pour en consulter le détail → Cette action crée et affiche une nouvelle activité.



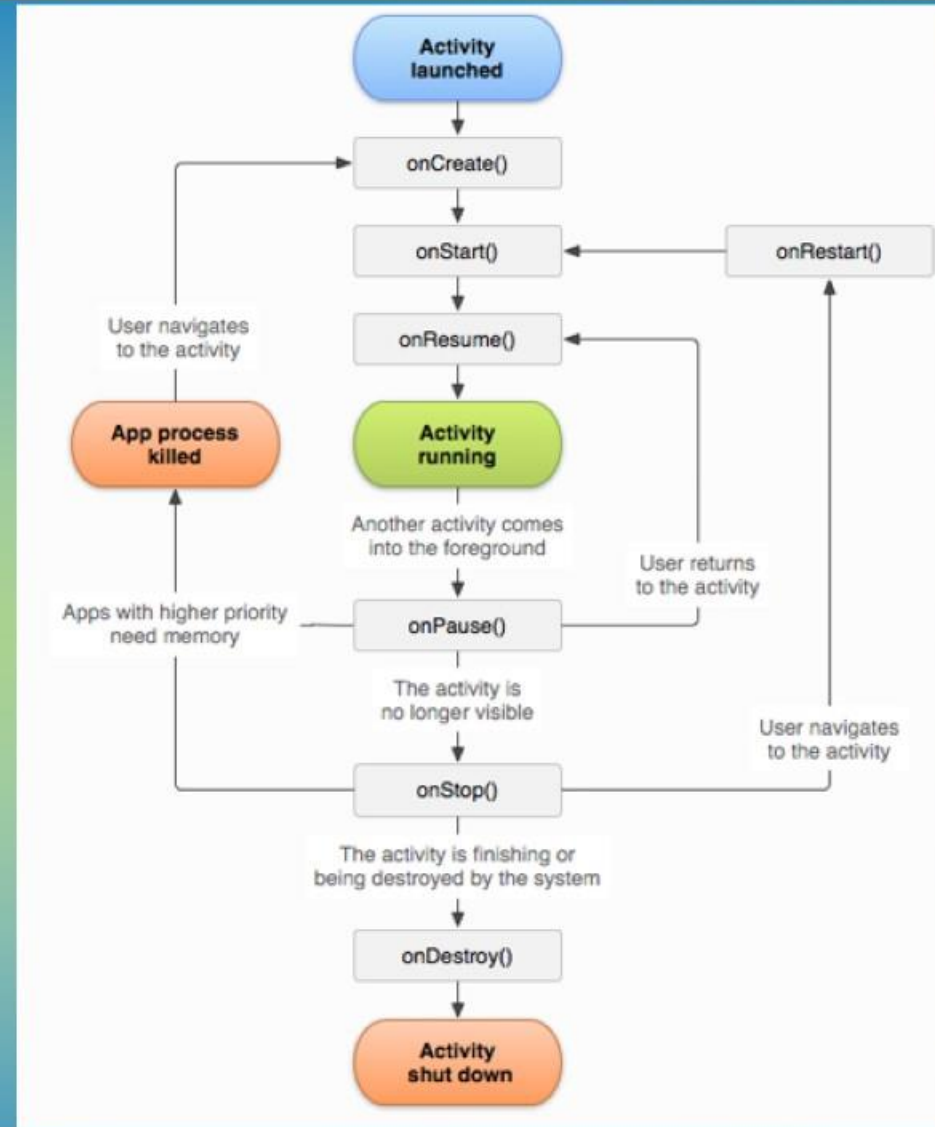




L'activité de **liste des mails** est **ARRÊTÉE** (`onPause` puis `onStop` sont appelées) → elle n'est plus au premier plan.

L'utilisateur appuie sur le **bouton Back** pour ramener la liste des mails.

L'activité doit être à **NOUVEAU AFFICHÉE** et ses méthodes `onRestart`, `onStart`, `onResume` sont appelées avant cela.

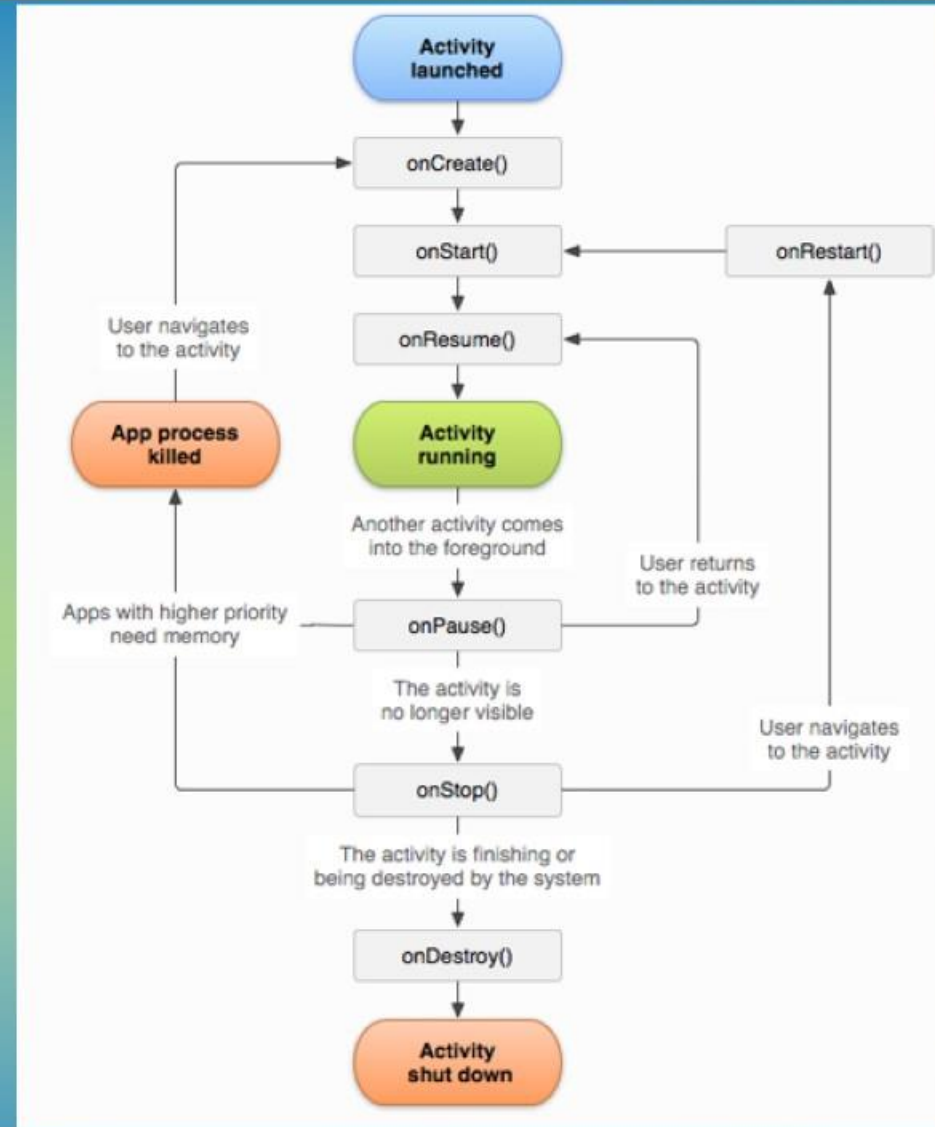




L'utilisateur **reçoit** un coup de téléphone et répond → Application de téléphonie en premier plan.

Activité de liste des mails est à nouveau **ARRÊTÉE** (**onPause**, **onStop**).

Communication téléphonique dure un certain temps → le système décide de **libérer** des ressources...

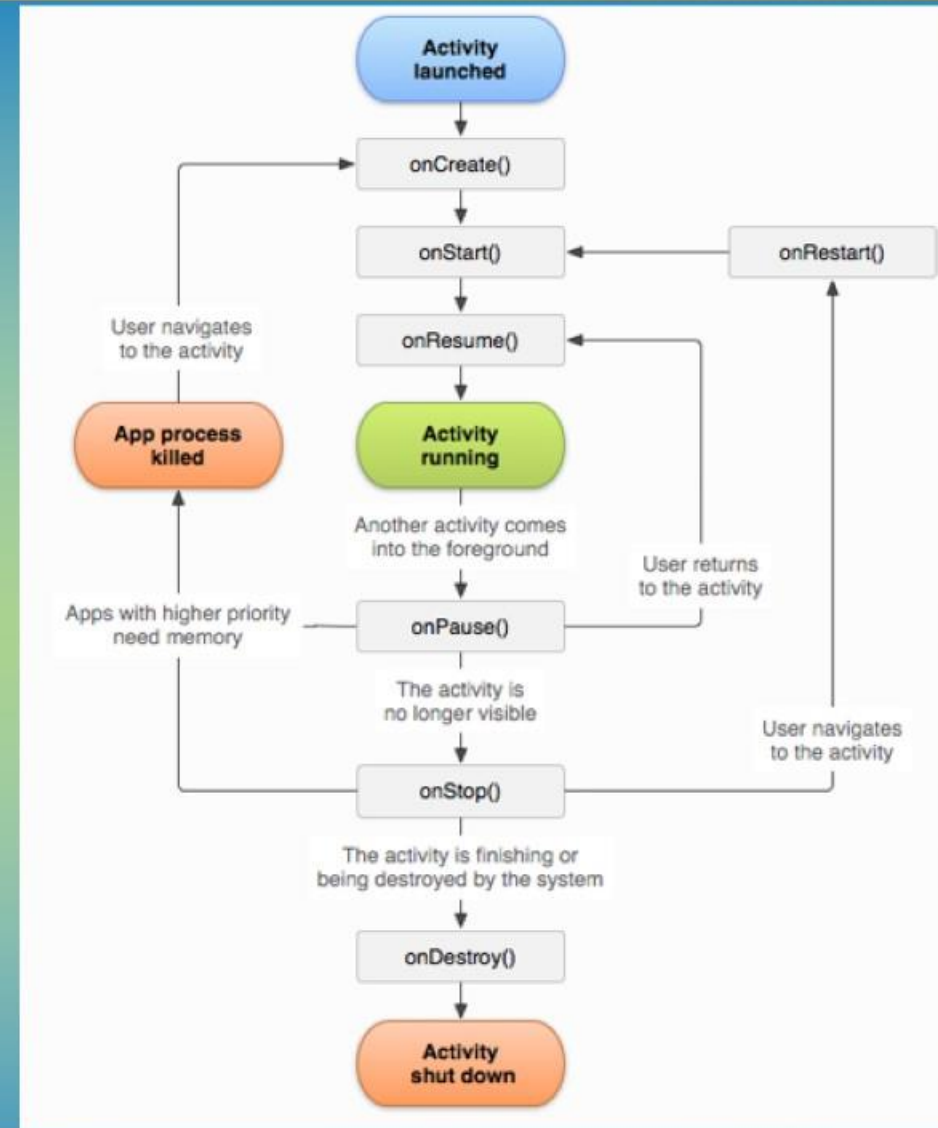




Lorsque l'utilisateur raccroche et revient à l'application de mails, l'activité est entièrement **RECRÉÉE** (`onCreate`, `onStart`, `onResume`).

L'utilisateur décide d'**ARRÊTER** l'application de mail.

Activité est alors complètement **DÉTRUITE** (`onDestroy`).







# Etats d'un cycle de vie





# onCreate

Elle est appelée lorsque votre activité est créée par le système. Généralement, les opérations effectuées dans cette méthode servent à mettre en place **l'interface graphique, à initialiser les variables, etc.**

C'est dans cette méthode que doivent être réalisés les traitements à exécuter **une seule fois pour toute la vie de l'activité.**

À ce stade, l'activité est créée, mais l'utilisateur **ne la voit pas encore** et ne peut pas interagir avec les différents éléments.





## onStart

Permet de savoir quand une activité va être rendue visible à l'utilisateur. **L'interface graphique devient visible à l'utilisateur**, mais il **ne peut pas encore interagir** avec les différents éléments.

Par exemple, vous pouvez faire des mises à jour de l'interface graphique avant affichage. Si votre activité joue une animation, cette méthode est une bonne candidate pour lancer automatiquement l'animation.







## onResume

L'activité devient **entièrement opérationnelle**. L'utilisateur peut utiliser l'application et cliquer sur les différents éléments graphiques. C'est le bon moment pour déclencher des opérations consommatrices de ressources (*activation de la caméra, ouverture de connexions réseaux, etc.*)

L'application **reste dans cet état tant qu'il n'y a pas d'interruption** : la réception d'un appel téléphonique, le démarrage d'une nouvelle activité, l'affichage d'une boîte de dialogue, etc.





# onPause

Une activité passe dans cet état par exemple **lorsqu'une AlertDialog est affichée** : l'activité sous-jacente est toujours visible, mais elle n'est pas au premier plan.

Cette méthode est **l'inverse** de la méthode onResume() : tout ce qui est initié dans onResume() doit être mis en pause dans cette méthode.

C'est **le moment de suspendre les opérations consommatrices de ressources** (*désactivation de la caméra, fermeture des connexions réseau, etc.*). Attention, une **activité en pause peut toujours être visible à l'utilisateur**. Vous ne devriez donc pas utiliser cette méthode pour modifier l'interface graphique ou stopper des animations.

Cette activité peut quand même être détruite par le système si ce dernier se trouve en manque de mémoire par exemple.







# onStop

Lorsqu'une nouvelle activité est démarrée, l'activité actuelle va se retrouver dans cet état. Elle n'est donc **plus du tout visible à l'utilisateur**.

Si votre activité a besoin de **sauvegarder des informations** (notamment sur disque) pour pouvoir redémarrer convenablement, vous pouvez effectuer ces traitements à ce moment-là. Vous pouvez utiliser cette redéfinition pour **stopper les opérations graphiques** (*animations, lecteur de vidéo, etc.*).







## onDestroy

Elle est appelée lors de la **mort de l'activité** (naturelle ou par le système).

Remarque : Quand une activité est dans un des états « onPause, onStop et onDestroy », elle est dans une situation où **elle peut être détruite pour libérer temporairement des ressources**. Cela ne signifie donc pas que l'activité a été simplement fermée par l'utilisateur. Parfois, l'urgence du système détruira l'activité sans même appeler onDestroy.

L'état « onPause » est le seul qui a la possibilité de s'exécuter complètement avant que l'activité ne soit détruite complètement. Il convient donc de **sauvegarder l'état de l'application dans cette méthode**.





## onRestart

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

L'appel à cette méthode se situe entre onStop et onStart. Cela permet plus facilement de faire la **différence entre le démarrage normal de l'activité et son redémarrage.**



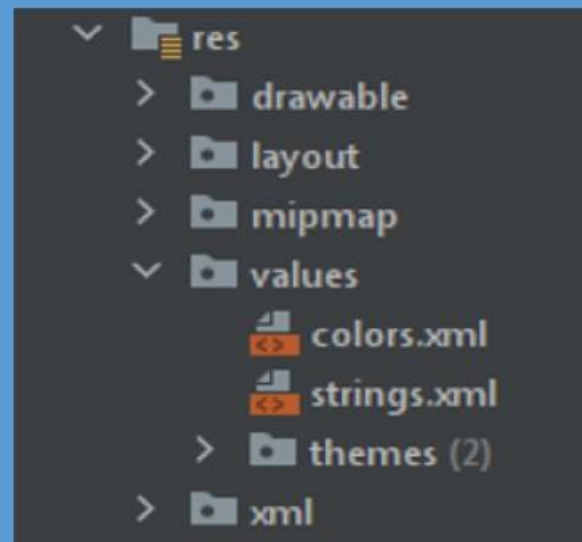


# Utilisation des ressources





Nous pouvons distinguer plusieurs types de ressources (*images, vidéo, audio, chaînes de caractères, etc.*) et toutes ces ressources se trouvent dans votre application sous le répertoire **res/**.





**res\drawable** : ce répertoire contient les ressources qui peuvent être dessinées. Parmi ces ressources, l'on peut citer les images (jpeg, gif, etc.), des formes génériques (cercles, carrés, etc.) définies dans des fichiers XML, etc.;

**res\layout** : les différentes vues nécessaires pour la mise en page ;

**res\values** : des fichiers XML qui contiennent des valeurs simples, telles que des chaînes de caractères, des entiers et des couleurs, les styles, etc. ;





# Chapitre 4

## Interfaces Graphiques et Widgets







Interagir avec l'utilisateur → **éléments graphiques** et des **widgets** (boutons, zones de saisie, menus déroulants, etc.)

Utilisons un dossier **layout** → contient **un fichier XML** que l'activité va charger.

Stocké dans le répertoire **res/layout** de votre projet.

S'il est **lié à une activité** → **suffixé par activity**, tout en **minuscules** et séparé par un **underscore**. Ex : MainActivity → activity\_main.xml.





## Editeur graphique

Android Studio ouvre l'éditeur en **mode Design**



vous pouvez placer et configurer les différents éléments graphiques avec votre souris



**générer automatiquement le contenu XML**





## Conteneurs

Afficher des éléments à l'écran



utiliser un **conteneur**



Un conteneur est un élément particulier permettant  
d'organiser les éléments qu'il contient entre eux



**LinearLayout** : positionner les éléments dans le sens horizontal ou vertical ;

**RelativeLayout** : permet de positionner les éléments les uns par rapport aux autres ;

**ConstraintLayout** : règles de positionnement beaucoup plus puissantes.







# Attributs

Chaque balise XML possède plusieurs attributs.

Deux attributs fondamentaux : **layout\_width** et **layout\_height**.

Permettent de déterminer comment afficher un élément au sein de son conteneur.

Valeurs possibles :

- **match\_parent** : Occuper le maximum d'espace disponible offert par son parent ;
- **wrap\_content** : Occuper que la place nécessaire à l'affichage de son contenu ;
- **0dp** : la taille de l'élément est définie par ses contraintes.





# Widgets





Un widget est un **élément de base**



**Afficher** du contenu à l'utilisateur

**Interagir** avec l'application

***Documentation officielle d'Android***







## **TextView**

Afficher une chaîne de caractères

## **EditText**

Ecrire des textes

## **Button**

Simple bouton (il s'agit en fait d'un TextView cliquable)





### CheckBox

Une case qui peut être dans deux états : cochée ou pas

### RadioButton et RadioGroup

Même principe que la CheckBox, à la différence que l'utilisateur ne peut cocher qu'une seule case





# Les évènements sur les widgets







# Gérer les interactions entre l'interface graphique et l'utilisateur

Plusieurs façons d'interagir avec une interface graphique  
Cliquer sur un bouton, entrer un texte, sélectionner une portion de  
texte, etc.

Ces interactions s'appellent des **évènements**





Pour pouvoir réagir à l'apparition d'un évènement, il faut utiliser un **objet** qui va détecter l'évènement et afin de vous permettre le traiter. Ce type d'objet s'appelle

### Les listeners

Vous oblige à redéfinir des méthodes et chaque méthode sera appelée au moment où se produira l'évènement associé





Enfin pour associer un listener à une vue  
on utilisera une méthode du type

**setOn[Evenement]Listener**

avec Evenement l'évènement concerné,

Ex : Pour détecter les clics sur un bouton on fera...







# Chapitre 5

## Menus et boites de dialogues





Un projet créé avec Android Studio utilise un thème qui affiche

**barre d'action + libellé de l'application**

Pour ajouter des boutons dans la barre d'action, vous devez  
commencer par créer  
un **menu** → ressource dans le répertoire **res**





Un **menu d'options** s'affiche lorsque l'utilisateur clique sur le bouton de menu de son appareil

Pour associer un menu à une activité, il faut surcharger la méthode **onCreateOptionsMenu**

Ce menu doit être peuplé avec des éléments

Sur ces éléments que cliquera l'utilisateur pour activer ou désactiver une fonctionnalité

Ces éléments s'appellent en XML des **<item>** et peuvent être personnalisés à l'aide de plusieurs attributs







Chaque **item** du menu possède un identifiant unique

Pour déclencher une action lorsque l'utilisateur clique sur un bouton du menu, il faut surcharger la méthode **onOptionsItemSelected**

Elle prend en paramètre l'**item** choisi dont il est possible de récupérer l'**ID** pour décider de l'action à réaliser





**Menu contextuel** s'affiche lorsque vous appuyez **longtemps** sur un élément de votre interface

Vous devez construire votre menu à partir de la méthode  
**onCreateContextMenu**





Pour créer une **boite de dialogue**, vous pouvez utiliser

La classe **AlertDialog** + la classe **AlertDialog.Builder**







Les boîtes de dialogue ont généralement un, deux ou trois boutons désignés sous

**terme de positif** (par exemple ok)

**neutre** (par exemple « plus tard »)

**négatif** (par exemple « annuler »)





Vous pouvez utiliser la classe **AlertDialog.Builder** pour définir le **titre**, le **message** et les **boutons** de la boite de dialogue

appeler ensuite la **méthode show** pour présenter la boite de dialogue



*Commence par faire le nécessaire, puis fait ce qu'il  
est possible de faire et tu réaliseras l'impossible  
sans t'en apercevoir*

