

# Développement d'applications Mobiles (Sous Android)

- Points Importants:  
N°1, N°6, N°7  
N°8, N°9

## CHAPITRE I: Introduction à l'univers d'android.

### 1- Définition d'android:

\* Android est une plateforme ouverte pour les appareils mobiles. Il s'agit d'un système d'exploitation complet qui comprend:

- Le Noyau Linux: c'est la base du SE, il gère les ressources matérielles du périphérique.

- Le middleware: Il fournit des services aux applications, tels que la gestion des contacts, la sécurité et la connectivité réseau.

- des applications: Elles constituent les programmes que les utilisateurs exécutent sur leur appareil, comme les jeux, les navigateurs web et les ressources sociales.

\* Types d'appareils mobiles: smartphones, tablettes, android TV, ereaders (liseuses électroniques), google glass.

### Avantages d'android:

- OpenSource: le code source d'android est disponible gratuitement, ce qui permet aux développeurs de créer des applications personnalisées et d'améliorer le SE.

- Extensible: ce qui signifie qu'il peut être adapté à différents types d'appareils.

- Large charte d'applications: il existe un grand nombre d'applications disponibles pour android.

## Avantages d'android:

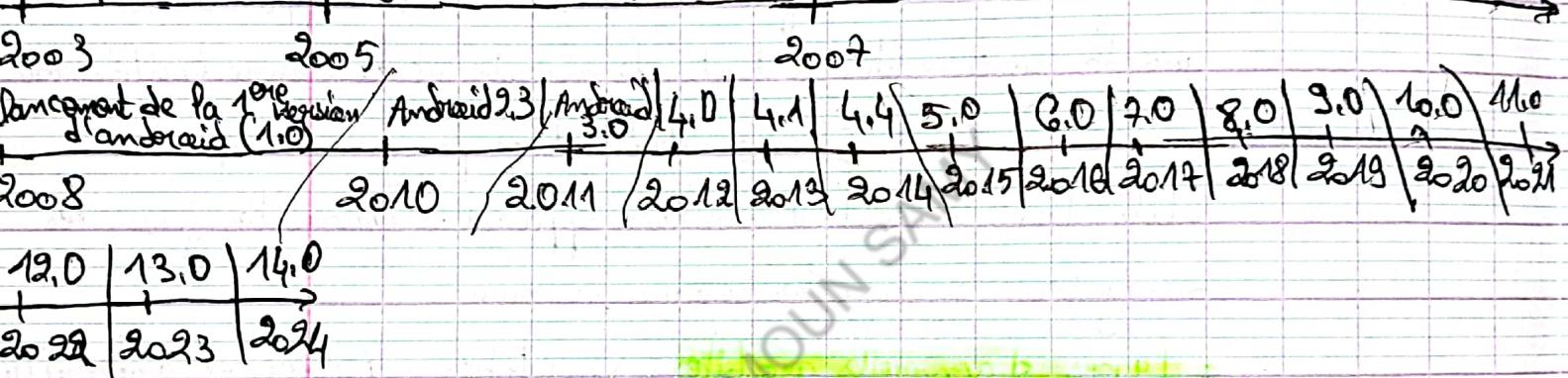
**Fragmentation:** la fragmentation est un problème sur android, car il existe de nombreuses versions différentes du SE. cela peut rendre difficile le développement d'applications compatibles avec tous les appareils android.

## 2- Historique :

Android est le SE mobile le plus populaire au monde, il est utilisé par des milliards de personnes dans le monde entier.

création de l'entreprise android

Google rachète l'entreprise android création de l'Open Hand Alliance (OHA)



## 3- Statistiques d'utilisation des Versions:

- **Version d'android:** est le numéro de version du système d'exploitation
- **Niveau d'API:** est le numéro qui identifie les fonctionnalités disponibles dans une version d'Android.
- **Distribution cumulée:** est le pourcentage d'appareils android qui utilisent une version d'android particulière ou une version ultérieure.
- Android 10 est la version la plus populaire avec une distribution cumulée de 99.5%.
- la 2ème: Android 11 avec 99.2%.

#### 4- Concurrents d'android:

- Android est le SE mobile le plus populaire au monde, avec une part de marché à 71,6% en 2022
- IOS est le 2<sup>eme</sup> SE mobile le plus populaire au monde, avec une part de marché de 28,2% en 2022
- les autres SE mobiles, tels que windows phone et blackberry OS ont une part de marché inférieure à 1%

#### 5- Applications Android sur Play Store:

- Le prix moyen des applications android a augmenté de manière constante de 2010 à 2022. (2010: 0,99\$ et en 2022: 3,49\$)

#### 6- Architecture d'android:

**Couche Applications:** les applications telles que home, contacts, phone, browser... (contient app native +)

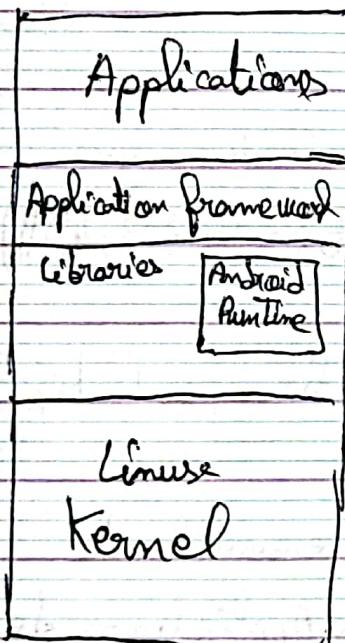
**Couche Application Framework:** elle fournit un ensemble de services et de gestionnaires qui permettent aux Applications de fonctionner de manière cohérente et efficace et voici certains composants de cette couche:

- Activity Manager: gère le cycle de vie des activités dans une application.
- Window Manager: gère les fenêtres affichées à l'écran, y compris leur placement et leur interaction.
- Content providers: gère l'accès aux données et les partages entre les app.
- View system: fournit les composants d'interface user de base.
- notification manager: gère les notifications système....
- package manager: gère l'installation, suppression et la mise à jour des app.
- Telephony manager: fournit les informations sur le réseau et l'état de ce.
- Resource manager: gère les ressources utilisées par les app (string, images)

③

- location manager: fournit des informations sur la localisation du périphérique.
- XMPP Service: gère la communication en temps réel via le protocole XMPP.
- Cache librairies: ensemble de bibliothèques logicielles utilisées par le SE et les apps pour fournir diverses fonctionnalités.
- surface manager: gère la composition de l'écran et l'affichage des surfaces graphiques.

- Media Framework: Fournit des fonctionnalités multimédia telles que la lecture audio et vidéo.



- SQLite: une BDD relationnelle intégrée pour le stockage de données.
- OpenGL ES: une API pour les graphiques 3D sur les appareils mobiles.
- FreeType: une bibliothèque pour le rendu de polices de caractères.
- WebKit: le moteur de rendu Web utilisé par le navigateur android.
- SEL: Simple Graphic Library une bibliothèque pour le rendu graphique simple.
- SSL: Secure Socket Layer pour sécuriser les communications réseau.
- **Android Runtime**: l'environnement d'exécution d'application utilisé par le SE android.
- Core libraries: Bibliothèques JAVA de base utilisées par les Apps android.
- Dalvik Virtual Machine: Machine Virtuelle spécifique à Android pour l'exécution des App.
- **Linus Kernel**: gère les pilotes matériels pour différents composants tels que l'écran, la caméra, le Bluetooth, ainsi que la gestion de la mémoire, du système de fichiers.

## 7. Architecture détaillée :

Lancement de l'app

Initialisation par l'Activity Manager

Création du processus zygote

Création du processus d'application.

- l'utilisateur lance une application en appuyant sur son icône
- lorsque l'application est lancée l'Activity manager reçoit l'autorisation de l'activité spécifique de l'application.
- l'Activity Manager communique avec le processus zygote qui est un processus parent spécial utilisé pour démarrer de nouveaux processus android de manière efficace
- le processus zygote est déjà en cours d'exécution en arrière-plan, prêt à créer de nouveaux processus d'application.
- le processus zygote reçoit la demande de l'Activity Manager pour créer un nouveau processus pour l'application.
- il crée alors un nouveau processus en chargeant la Machine Virtuelle ART (ou Dalvik avant android 5.0) et d'autres composants.

## 8. La chaîne de compilation sous android :

1- Compilation du code source JAVA en bytecode JVM : les programmes pour android sont généralement écrits en JAVA et compilés en bytecode pour la Machine virtuelle JAVA (JVM)

2- Conversion en bytecode DALVIK/ART : Le bytecode JVM est ensuite converti en bytecode Dalvik ou ART qui est le format d'exécution pour les applications Android. Cette conversion est effectuée à l'aide de l'outil dx (Dalvik Exchange).

3- Génération des fichiers .dex : Le bytecode converti est stocké dans des fichiers au format .dex (Dalvik Executable) pour l'exécution sur la Machine virtuelle Dalvik ou ART.

## 9. Les Composants d'une App Android:

Application = ensemble de Composants Android.

### Activites (Activities)

- offre une interface utilisateur
- généralement représente un seul écran
- étend la classe Activity

### Services

- Pas d'interface utilisateur
- s'exécute en arrière plan
- étend la classe Service.

### Défiseurs (Broadcast receiver/intent)

- reçoit et répond aux broadcast intents
- Pas d'UI mais peut démarquer une Activity
- Etend la classe BroadcastReceiver

### Contenus (Content Provider)

- Permet à une application accessible à d'autres app.
- Données sauvegardées dans la BDD SQLite
- Etend la classe Content Provider

## CHAPITRE II: Outils de développement Android

1- La pile d'outils: la pile d'outils Android fait référence à l'ensemble des logiciels, frameworks et bibliothèques fournis par Google pour le développement d'applications Android. Cette pile d'outils est essentielle pour les développeurs Android, car elle leur permet de créer, tester, débugger et déployer des applications pour les appareils Android.



### 2- Le JDK/Eclipse:

JDK/JRE : pour développer une App android (base) sur le langage JAVA. il est nécessaire d'avoir le JDK/JRE

JDK (JAVA Development Kit) : est un ensemble d'outils essentiels pour développer des Apps JAVA et android. Elle comprend des exécutables tels que la Machine virtuelle JAVA (JVM), le compilateur Java, et un ensemble de classes de base avec des fonctionnalités telles que l'interface utilisateur, les conteneurs, la gestion des threads ...

- JRE (JAVA Runtime Environment): est nécessaire pour exécuter des applications Java. Il inclut principalement la Machine Virtuelle JAVA (JVM) qui est responsable de l'exécution du bytecode JAVA sur une plateforme donnée.
- Android Studio: environnement de développement (IDE)

3- SDK: est un ensemble complet d'outils, de documentation, d'exemples de code et d'APIs fournis par Google pour faciliter le développement d'applications Android.

- APIs: ensemble de classes regroupant des fonctionnalités mises à disposition des développeurs.
- Documentation: contient 2 parties : une pour le guide du développeur et l'autre expliquant l'utilisation des APIs.
- Les exemples: code prêt à l'emploi pour illustrer l'utilisation des fonctionnalités de la plateforme.
- Le SDK Android comprend une gamme d'outils de développement indispensables pour créer des applications Android:
  - aapt (Android asset packaging tool): son rôle majeur est de compiler les ressources des app android.
  - AVD (Android Virtual Device): permet de créer et de gérer des emulateurs Android pour tester des applications sur différentes configurations d'appareils.
  - ADB (Android Debug Bridge): est un outil permettant la communication avec l'emulateur.

#### 4- Gradle :

- Automatise et simplifie le processus de Construction des Applications Android
- responsible de la construction du Projet (Compilation des sources et des ressources)
- Permet la Configuration des projets; definir divers paramètres, options et variables dans les fichiers de Configuration gradle.
- Permet la gestion automatique de dépendance: telle que des bibliothèques tierces, modules...)

#### 5- Processus de Construction d'une Application Android :

- Compilation et packaging: le code source JAVA compilé en fichier .dex qui sont ensuite regroupés avec l'application (fichiers .arsc, images) pour créer un package android (.apk)
- Signature: l'application est signée numériquement pour garantir son authenticité et empêcher la falsification.
- Déploiement: l'app peut être installée sur l'appareil Android via un câble USB, le play store ou....



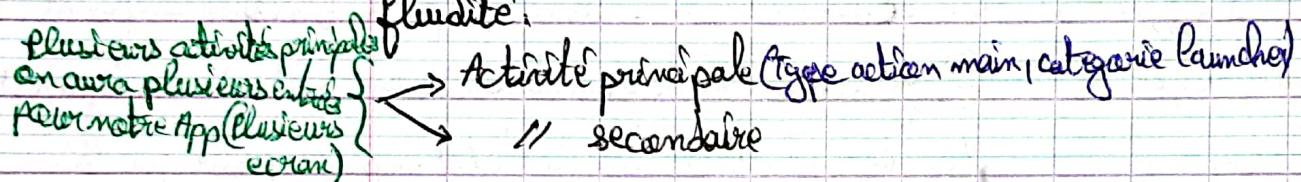
## CHAPITRE 3: Néotion Activité

### 1- Les Composants d'une App Android:

Activité, services, intent, broadcast, content provider.

Activité offre une interface, un cycle de vie qui gère par l'activité manger (create et destroy), étend la classe activity.

activity manger manager : gère le cycle de vie pour mettre en place la gestion des ressources (sans la gestion il y aurait pas la fluidité).



### 2- Etats d'activité:

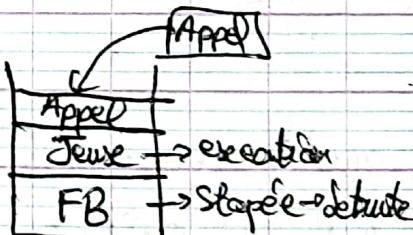
- Act en état d'exécution : l'activity manager gère les act en utilisant le cycle de vie + pile d'exécution.

une fois que l'appel arrive il sera en haut

de la pile d'exécution et il devient une act

en exécution, seule sera stoppé une fois act

appel sera terminé il passe à l'état stoppé ensuite détruit et le pile d'exécution  
jeu reprendra son exécution.



### 3- Première Activité:

- L'activité étend la classe Activity (Android.app.Activity)

- L'activité implemente la Méthode : OnCreate(): initialiser, SetContentView()

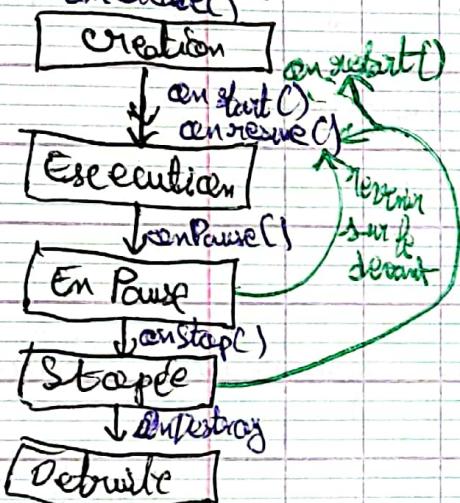
```
import android.app.Activity;
import android.os.Bundle;
public class ExempleActivite extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.interface);
    }
}
```

La méthode `setContentView(R.layout.interface)` est utilisée pour définir le contenu de l'interface utilisée par le `Placeholder`, c'est à dire `XML interface.xml`.

La classe `ExempleActivite` étend la classe `Activity` ce qui rend l'activité utilisable en tant qu'activité Android dans l'application. La méthode `onCreate()` est une méthode de rappel (callback) du cycle de vie de l'activité. Elle est appelée lorsque l'activité est créée pour la première fois. A l'intérieur de `onCreate()`, il y a un appel à `super.onCreate(savedInstanceState)`. Cela garantit que le comportement par défaut de la méthode `onCreate()` de la classe parente `Activity` est également exécuté. Elle charge la disposition (layout) définie dans le fichier.

## 4- Cycle d'une activité :

onCreate, avant tout sa flächette doit être créé par une Méthode onCreate()



- ① Scénario normal : l'acteur app → utilisation → ferme
  - ② Je suis à l'état exécution et une boîte de dialogue est lancée, l'act va passer à l'état en pause pour que la boîte de dialogue s'exécute une fois terminée, l'act revient à l'état exécution grâce à `@OnResume()`
  - ③ Si on reçoit un appel pendant qu'un act est en cours d'exécution l'appel sera enregistré, en statut et il va

mettre en exécution l'appel une fois terminer, l'act principal reprend grace à `onRestart`, `onStart()`, `onResume()` (Exemple : recevoir un Appel pendant <sup>qu'un jeu</sup> ~~qu'il est~~ la partie).

- 4) Si une Act a besoin ~~d'un~~ de ressources qu'on passe de pas. si c'est un problème de montrer l'activité manager va détruire le processus qui héberge l'activité qui est soit à l'état stoppé ou en pause pour qu'il puisse lancer l'activité qui lui demande beaucoup de ressources. (ordre : supprime les processus vide ensuite celle qui héberge l'act stoppé et en pause)

5) lors de la rotation de l'écran faut faire un cycle en laps de temps (exemple : jouer en mode portrait ensuite on fait la rotation, il va détruire l'act et la scene (onPause, onStop, onDestory, onCreate...))

2

## + Les Méthodes

**onCreate()**: l'activité n'est pas dans la file. appelée à la création, son rôle est de faire des initialisations (interface graphique) il a un paramètre de type bundle (garde l'état des informations) elle se termine avec succès alors la méthode **onStart()** est appelée

**onStart()**: l'activité devient visible à l'utilisateur mais n'a pas encore le focus. (Si l'act a le focus alors onResume() -> onStop())

**onResume()**: l'act devient active et a le focus de l'utilisateur. c'est le moment où vous pouvez démarrer des animations, des mises à jour en temps réel et reprendre des opérations en pause (utilisation aps, elle se termine normalement alors l'act est en execution)

**onPause()**: l'act n'a plus le focus mais reste visible à l'utilisateur. Vous devez mettre en pause ou libérer des ressources inutiles ici.

Appelée lors d'une autre Act vient au  
Branier Plan, l'act est détruit,

**onStop()**: l'act n'est plus visible à l'utilisateur. Vous pouvez arrêter des opérations consommatoires de ressources ici.

Similaire à onCreate **onRestart()**: l'act est redémarré après avoir été mise en pause. Cette méthode est appelée avant que l'act ne devienne visible à nouveau.

peut être appelée lorsque: l'user Tape sur le bouton retour, faire appel à la méthode a besoin de plus de ressources.

**onDestroy()**: l'act est détruit et libère toutes ses ressources. Cela se produit lorsque l'utilisateur quitte l'act ou lorsque le système

Exemple de cycle de vie d'une Act : page 50.

- Attribut tag: une chaîne de caractères "tag" est définie pour identifier les messages de journalisation dans le Méthodes du cycle de vie.

- log.d(): méthode de la classe "log" dans Android utilisée pour enregistrer des messages de débogage dans le système de journalisation de l'app. Elle prend 2 arguments (tag, "message"). log.d(tag, "le message")

Pour filtrer les messages selon leur tag.

- super.nom\_de\_la\_methode(); dans chaque méthode surcharge, un appel à la méthode parente correspondante est effectué pour garantir que le comportement par défaut de l'activité est également exécuté.

## 5. Sauvegarde de l'état (persistance):

Pour éviter la perte de données, notamment si le système détruit l'activité, car qu'il a besoin de mémoire, il faut alors les sauvegarder, plusieurs options sont offertes : bundle, BD, fichiers. Lors de la rotation, le système détruit l'application exemple seu en mode paysage puis portrait, le score ~~va~~ va pas être supprimé, on dit que les données ont été sauvegardées temporairement dans le bundle.

Bundle : tableau associatif clé / valeur type, il offre des méthodes pour stocker les données : putBoolean, putInt, putParcelable, putSerializable et aussi des méthodes pour récupérer les données : getBoolean, getInt, getParcelable.

Parcelable : décrit comment stocker un objet complexe dans un bundle elle est appellée juste avant enfin `onSaveInstanceState(Bundle monBundle)`: sauvegarder les données temporairement.

elle est appellée juste avant enfin `onRestoreInstanceState(Bundle monBundle)`: juste après la création

explication exemple page 54: `onSaveInstanceState(Bundle outstate)`



## Chapitre 4: les interfaces graphiques sous android

### 1/Définitions:

#### Interface Graphique

- **Définition :** Une interface graphique est un moyen de communication entre un utilisateur et une application. Elle contient un ensemble de composants graphiques qui permettent l'interaction.

#### Déclaration de l'Interface

- **Méthodes de déclaration :**
  - **En XML :** La structure de l'interface est définie dans des fichiers XML.
  - **Dans le code Java de l'activité :** L'interface est construite directement dans le code Java de l'application.

### 2/Explication étape par étape du code Java et XML:

#### Composants : XML VS Java

```
public class ExempleActivity extends Activity {  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        LinearLayout conteneur = new LinearLayout(this);  
        conteneur.setOrientation(LinearLayout.VERTICAL);  
        TextView texte = new TextView(this);  
        texte.setText(" une interface par un programme ");  
        conteneur.addView(texte);  
        setContentView(conteneur);  
    }  
}
```

JAVA

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="Une interface en XML"  
        android:id="@+id/text" />  
</LinearLayout>
```

XML

#### Code Java:

1. **Déclaration de la classe ExempleActivity :**
  - La classe **ExempleActivity** étend **Activity**, ce qui signifie qu'elle est une sous-classe d'**Activity** et peut définir son propre comportement pour les méthodes d'**Activity**.
2. **Méthode onCreate :**

- La méthode `onCreate` est appelée lorsque l'activité est créée. Elle initialise l'interface utilisateur de l'activité.
- `super.onCreate(savedInstanceState);` appelle la méthode `onCreate` de la classe parente pour effectuer toute configuration initiale propre à `Activity`.

### 3. Création du `LinearLayout` :

- `LinearLayout conteneur = new LinearLayout(this);` crée un nouveau `LinearLayout`, qui est un type de conteneur qui aligne ses enfants en une seule direction (verticale ou horizontale).
- `conteneur.setOrientation(LinearLayout.VERTICAL);` définit l'orientation du `LinearLayout` sur verticale.

### 4. Création du `TextView` :

- `TextView texte = new TextView(this);` crée un nouveau `TextView`.
- `texte.setText(" une interface par un programme ");` définit le texte affiché par le `TextView`.

### 5. Ajout du `TextView` au `LinearLayout` :

- `conteneur.addView(texte);` ajoute le `TextView` au `LinearLayout`.

### 6. Définir le `LinearLayout` comme contenu de l'activité :

- `setContentView(conteneur);` définit le `LinearLayout` comme l'interface utilisateur de l'activité.

## Code XML:

### 1. Déclaration de la version XML :

- `<?xml version="1.0" encoding="utf-8"?>` spécifie la version XML et l'encodage utilisé.

### 2. Déclaration du `LinearLayout` :

- `<LinearLayout>` est l'élément racine du layout.
- `xmlns:android="http://schemas.android.com/apk/res/android"` définit l'espace de noms Android, nécessaire pour utiliser les attributs Android.
- `android:orientation="vertical"` définit l'orientation du `LinearLayout` sur verticale.
- `android:layout_width="match_parent"` et `android:layout_height="match_parent"` définissent la largeur et la hauteur du `LinearLayout` pour qu'il occupe toute la largeur et la hauteur de son parent.

### 3. Déclaration du `TextView` :

- `<TextView>` définit un composant de texte à afficher.

- `android:layout_width="wrap_content"` et `android:layout_height="wrap_content"` définissent la largeur et la hauteur du `TextView` pour qu'ils s'ajustent à son contenu.
- `android:text="Une interface en XML"` définit le texte affiché par le `TextView`.
- `android:id="@+id/text"` donne un identifiant unique au `TextView`, permettant de le référencer dans le code Java si nécessaire.

## Résumé Comparatif

- **Java :**
  - Dynamique : L'interface est construite à l'exécution.
  - Flexible : Permet de modifier l'interface en fonction de la logique du programme.
  - Utilisé pour : Interface dont la structure dépend de la logique dynamique du programme.
- **XML :**
  - Statique : L'interface est définie avant l'exécution.
  - Déclaratif : Sépare la logique de l'interface de l'application.
  - Utilisé pour : Interface fixe, facilitant la maintenance et la réutilisation.

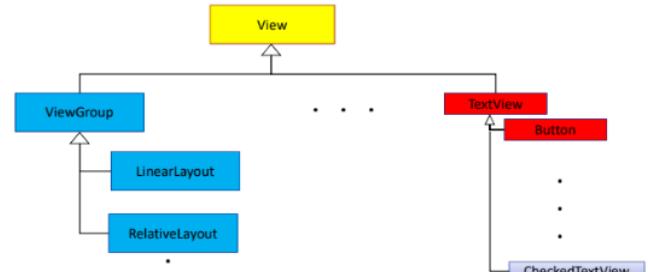
## 3/Les composants graphiques:

### Catégories de Composants Graphiques

#### 1. `android.view.View`

- **Description :** Composants individuels affichant du contenu ou recevant des interactions de l'utilisateur (ex. `TextView`, `Button`).

Les composants graphiques



#### 2. `android.view.ViewGroup`

- **Description :** Conteneurs regroupant plusieurs `View` ou `ViewGroup`.
- **Tâches :**
  - **Regroupement** : Regroupe un ensemble de composants enfants.
  - **Disposition** : Gère l'organisation et le positionnement des composants enfants, similaire au `LayoutManager` en AWT (Abstract Window Toolkit).

## 4/La classe view:

La classe **View** est la classe de base pour tous les composants graphiques en Android.  
Voici les principales propriétés de cette classe :

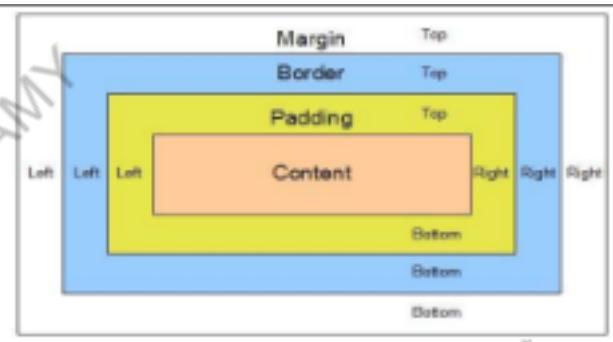
- **android:id** : Assigner un identificateur (ex. `@+id/mon_id`) au composant.
- **android:layout\_width** : La largeur du composant.
- **android:layout\_height** : La hauteur du composant.
- **android:visibility** : Contrôle la visibilité initiale du composant (visible, invisible, ou gone).
- **android:background** : Spécifie la couleur de fond au format RGB ou une image.

### Propriétés de Marges et de Padding

- **android:paddingTop, android:paddingBottom** : Espace intérieur en haut et en bas du composant.
- **android:marginTop, android:marginRight** : Espace extérieur en haut et à droite du composant.

### Schéma de Layout

- **Content** : Contenu principal du composant.
- **Padding** : Espace intérieur autour du contenu.
- **Border** : Bordure autour du padding.
- **Margin** : Espace extérieur autour de la bordure.



### 5/ Explication du Code Java et XML de l'exemple:

```
public class ExempleActivity extends Activity {  
    public TextView text;  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.vue);  
  
        text=(TextView)findViewById(R.id.text1);  
    }  
}
```

```
public final class R {  
    public static final class layout {  
        public static final int vue=0x7f030000;  
    }  
    public static final class id {  
        public static final int text1=0x7f070000;  
    }  
}
```

Code JAVA

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="Une interface en XML"  
        android:id="@+id/text1" />  
</LinearLayout>
```

Interface XML

Code Java:

1. **Déclaration de la classe `ExempleActivity` :**
  - La classe `ExempleActivity` étend `Activity`, ce qui signifie qu'elle hérite des fonctionnalités de la classe `Activity` et peut définir ses propres comportements.
2. **Déclaration de l'attribut `TextView` :**
  - `public TextView text;` : Déclare un attribut `text` de type `TextView` qui sera utilisé pour référencer le composant `TextView` dans le layout XML.
3. **Méthode `onCreate` :**
  - `@Override` : Indique que cette méthode redéfinit une méthode de la classe parente.
  - `protected void onCreate(Bundle savedInstanceState)` : Méthode appelée lorsque l'activité est créée. Elle initialise l'interface utilisateur.
  - `super.onCreate(savedInstanceState);` : Appelle la méthode `onCreate` de la classe parente pour effectuer toute configuration initiale propre à `Activity`.
  - `setContentView(R.layout.vue);` : Définit le fichier XML `vue` comme interface utilisateur pour cette activité. `R.layout.vue` fait référence à la ressource layout définie dans le fichier XML `vue`.
  - `text = (TextView) findViewById(R.id.text1);` : Trouve le `TextView` dans le layout avec l'identifiant `text1` et l'assigne à l'attribut `text`.
4. **Classe R :**
  - **R est une classe générée automatiquement qui contient les identifiants de toutes les ressources utilisées dans l'application.**
  - **Sous-classe layout :**
    - i. `public static final int vue = 0x7f030000;` : Identifiant unique pour le layout `vue`.
  - **Sous-classe id :**
    - i. `public static final int text1 = 0x7f070000;` : Identifiant unique pour le `TextView` avec l'ID `text1`.

#### Code XML:

1. **Déclaration de la version XML :**
  - `<?xml version="1.0" encoding="utf-8"?>` : Spécifie la version XML et l'encodage utilisé.
2. **Déclaration du `LinearLayout` :**
  - `<LinearLayout`  
 `xmlns:android="http://schemas.android.com/apk/res/android"` : Élément racine de ce layout, déclarant l'espace de noms Android nécessaire pour utiliser les attributs Android.

- `android:layout_width="match_parent"` : Le `LinearLayout` occupe toute la largeur du parent.
- `android:layout_height="match_parent"` : Le `LinearLayout` occupe toute la hauteur du parent.

### 3. Déclaration du `TextView` :

- `<TextView` : Déclare un composant de texte.
- `android:layout_width="wrap_content"` : La largeur du `TextView` s'ajuste à son contenu.
- `android:layout_height="wrap_content"` : La hauteur du `TextView` s'ajuste à son contenu.
- `android:text="Une interface en XML"` : Définit le texte affiché par le `TextView`.
- `android:id="@+id/text1"` : Assigne un identifiant unique `text1` à ce `TextView`.

## Résumé

- **Java :**
  - L'activité `ExempleActivity` charge le layout `vue`.
  - Un `TextView` est référencé par son ID `text1` dans le code Java.
- **XML :**
  - Le layout `vue` est défini avec un `LinearLayout` contenant un `TextView`.
  - Le `TextView` a une largeur et une hauteur ajustées à son contenu et affiche le texte "Une interface en XML" avec l'ID `text1`.

## 6/ Les conteneurs:

- **ViewGroup :**

Un `ViewGroup`, sous-classe de `View`, est un conteneur qui regroupe et dispose des composants (`View`) en utilisant un gestionnaire de disposition (Layout Manager). Chaque vue dans un `ViewGroup` doit spécifier `android:layout_height` et `android:layout_width`, avec des valeurs comme `match_parent`, `wrap_content`, ou une valeur exacte (dp, sp, px, etc.). Les conteneurs prédéfinis dans Android incluent `LinearLayout`, `RelativeLayout`, `TableLayout`, `FrameLayout`, et `AbsoluteLayout`, et un conteneur peut être imbriqué à l'intérieur d'un autre.

- **LinearLayout :**

Le `LinearLayout` aligne les composants soit de haut en bas soit de gauche à droite, selon la propriété `android:orientation` qui peut prendre les valeurs `Vertical` ou `Horizontal` (par défaut), ou en utilisant la méthode `setOrientation(int orientation)` avec les constantes `HORIZONTAL` ou `VERTICAL`. De plus, le `LinearLayout` comprend deux attributs supplémentaires : `android:layout_gravity`

(ou `setGravity(int)`) pour spécifier l'alignement des composants à l'intérieur du layout, et `android:layout_weight` (ou `setWeight(int)`) pour définir la répartition des espaces disponibles entre les composants lorsque `android:layout_width` ou `android:layout_height` est défini sur `0dp` (pour `LinearLayout`) ou `match_parent` (pour `RelativeLayout`).



- `<LinearLayout>` : Conteneur organisant les éléments de manière linéaire.
  - `android:orientation="horizontal"` : Les éléments sont disposés horizontalement.
  - `android:layout_width="match_parent"` : La largeur du LinearLayout correspond à celle de son parent.
  - `android:layout_height=` : La hauteur du LinearLayout est non définie dans le code donné.
- `<Button>` : Éléments de bouton permettant à l'utilisateur d'interagir.
  - `android:layout_width="wrap_content"` : La largeur du bouton s'ajuste selon son contenu.
  - `android:layout_height="wrap_content"` : La hauteur du bouton s'ajuste également selon son contenu.
  - `android:text="Bouton X"` : Texte affiché sur le bouton.
  - `android:id="@+id/BoutonX"` : Identifiant unique du bouton.
  - `android:layout_gravity="center"` : Centre le bouton horizontalement dans le LinearLayout.
  - `android:layout_weight="1"` : Donne un poids relatif au bouton par rapport aux autres éléments pour le dimensionnement dans le LinearLayout.

- **RelativeLayout:**

Le **RelativeLayout** est un conteneur de mise en page dans Android qui permet d'aligner les composants par rapport au conteneur parent et/ou par rapport à d'autres composants frères. Voici un résumé des propriétés couramment utilisées par ce conteneur :

- **android:layout\_alignParentRight** : Aligne le bord droit du composant avec le bord droit du conteneur parent.
- **android:layout\_alignParentTop** : Aligne le bord supérieur du composant avec le bord supérieur du conteneur parent.
- **android:layout\_toRightOf** : Place le composant à la droite d'un autre composant spécifié.
- **android:layout\_above** : Place le composant au-dessus d'un autre composant spécifié.
- **android:layout\_alignLeft** : Aligne le bord gauche du composant avec le bord gauche d'un autre composant spécifié.
- **android:layout\_alignBaseline** : Aligne la ligne de base du composant avec la ligne de base d'un autre composant spécifié.

## RelativeLayout: exemple



15

- **TableLayout:**

Le TableLayout est un conteneur de mise en page dans Android qui organise les composants sous forme d'une matrice, avec des lignes et des colonnes. Voici un résumé des caractéristiques principales de ce conteneur :

- Chaque ligne de la table est déclarée à l'intérieur de l'élément `<TableRow>`.
- Propriétés couramment utilisées par ce conteneur :
  - `android:layout_column` : Spécifie la colonne dans laquelle le composant doit être placé dans la ligne. Utile pour définir la position des composants dans une ligne.
  - `android:layout_span` : Permet à un composant de fusionner plusieurs colonnes adjacentes en une seule cellule. Cela permet de créer des composants qui couvrent plusieurs colonnes dans une seule ligne.
  - `android:stretchColumns` : Définit les colonnes qui doivent être étirées pour occuper tout l'espace horizontal disponible. Les colonnes spécifiées vont s'étirer pour remplir l'espace disponible.
  - `android:shrinkColumns` : Définit les colonnes qui peuvent être rétrécies si nécessaire pour s'adapter au contenu des cellules. Les colonnes spécifiées rétréciront pour éviter que le contenu ne soit coupé.
  - `android:collapseColumns` : Indique les colonnes qui peuvent être réduites à zéro largeur si nécessaire. Les colonnes spécifiées peuvent être réduites à zéro largeur si l'espace disponible est insuffisant.

## TableLayout: exemple

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/tableLayout" >

    <TableRow android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:id="@+id/firstRow">
        <Button android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button11" />
        <Button android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button12" />
        <Button android:id="@+id/button3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button13" />
    </TableRow>
    <TableRow android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/secondRow">
        <Button android:id="@+id/button4"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="0"
            android:layout_span="2"
            android:text="Button212" />
    </TableRow>
</TableLayout>
```



A.Hammache

17

## 7/contrôle:

composants ou views,c'est les éléments de base utilisés pour construire les interfaces user dans les applications.

- **Caractéristique et responsabilités des composants :**

1/Surface rectangulaire à l'écran:les composants sont des surfaces rectangulaires qui occupent une zone sur l'écran.

2/Responsable de dessin:chaque composant est responsable du rendu de son contenu visuel.Cela implique généralement la mise en place des éléments graphiques et leur affichage à l'écran.

3/Responsable de gestion d'événement:tels que les clics, les glissements, les touches pressées, etc. Ils peuvent détecter et répondre aux interactions de l'utilisateur, ce qui permet d'ajouter une interactivité à l'application.

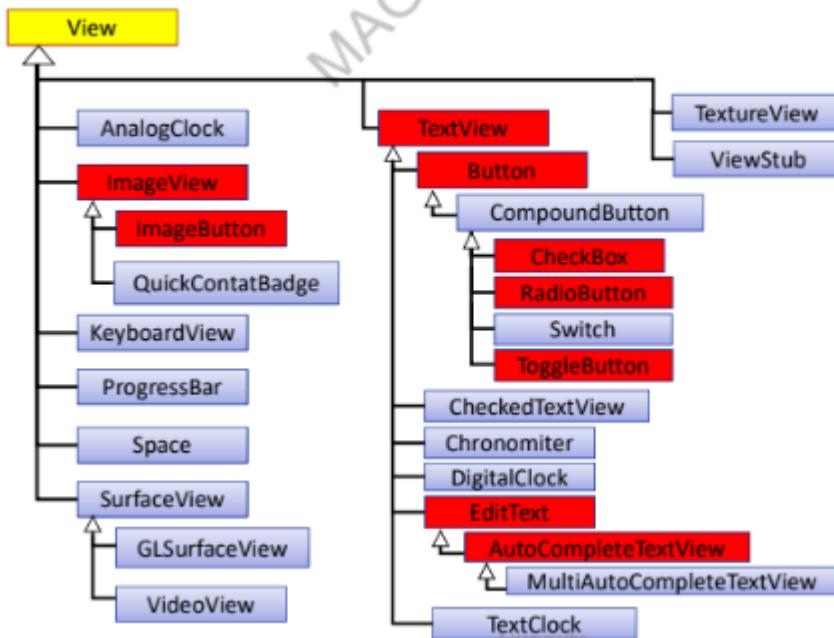
Exemple de composants:

google map et web view(permert d'afficher une carte google en chargeant une page web).

widgets:composants interactifs,autonomes qui peuvent être ajouté à la page d'accueil pour fournir des fonctionnalités.

- **Hiérarchie des composants:**

### Hiérarchie des composants (contrôle)



## 8/Label(TextView):

- Appelé aussi **Label** est une zone de texte non modifiable

Attributs importants	Méthodes Java
android :text : texte du label	setText(CharSequence)
android:textStyle : (bold, italic , bold_italic)	setTextStyle(Style)
android:textSize : (Taille en sp)	setTextSize(float)
android:textColor : ( en RGB : #FF0000)	setTextColor(ColorStateList)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:text= "Exemple de TextView"
        android:layout_width="wrap_content "
        android:layout_height="wrap_content"

        android:textStyle="bold"
        android:textColor="#FF0000"
        android:textSize="30sp"/>
</LinearLayout>
```

A.Hammache



20

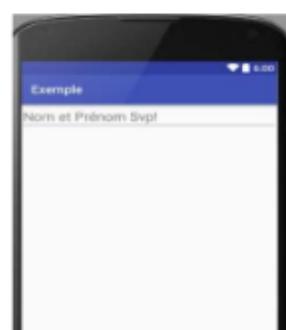
## 9/Champ de saisie(EditText):

- Permet à l'utilisateur de saisir un texte, représenté par **EditText**
- C'est une sous classe **TextView**

Attributs	Signification
android:autoText	Indique si le champ effectue une correction automatique de l'orthographe
android:capitalize	Mettre la première lettre en majuscule
android:digits	Le champ n'accepte que certains chiffres
android:singleLine	Indique si le champ est sur une seule ligne ou plusieurs
android:password	Contrôle la visibilité du champ
android:phoneNumber	Formater le champ pour des numéros de téléphone

```
<EditText
    android:id="@+id/txtUserName"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="22sp"
    android:autoText="true"
    android:capitalize="words"
    android:hint="Nom et Prénom Svp!<     />
```

A.Hammache

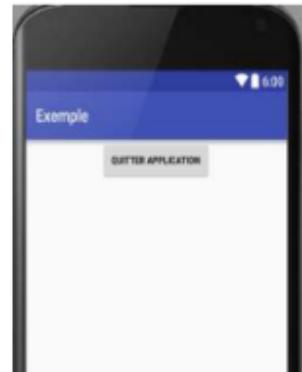


21

## 10/Bouton(Button):

- Un bouton permet la simulation de l'action de clic sur une interface
- Hérite de la classe TextView

```
<Button  
    android:id="@+id	btnExit"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Quitter Application"  
    android:textSize="24sp"  
    android:textStyle="bold"  
    android:gravity="center"  
  
</Button>
```



A.Hammache

22

## 11/CompoundButton:

- CheckBox:

- Est un bouton avec deux états (**checked/unchecked**)
- Les composants : **CheckBox**, RadioButton, Switch et ToggleButton

<b>isChecked()</b>	pour savoir si la case est cochée
<b>setChecked()</b>	pour forcer la case dans l'état coché ou décoché
<b>toggle()</b>	pour inverser l'état de la case

```
<CheckBox  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/buttonCheck"  
    android:text="CheckBox"  
    android:checked="true">  
</CheckBox>
```



A.Hammache

23

- **RadioButton :**

- Un bouton Radio est généralement placé dans un **RadioGroup** ( où un seul bouton est sélectionné à un instant donné)
- Hérite de la classe CompoundButton

```
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/buttonRadio1"
        android:text="Button Radio 1"
        android:checked="true"/>
    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/buttonRadio2"
        android:text="Button Radio 2"
        android:checked="false"/>
</RadioGroup>
```

A.Hammache



24

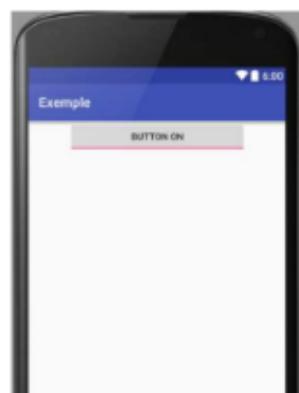
- **ToggleButton :**

Un bouton basculant (ToggleButton en anglais) est un élément d'interface utilisateur dans Android qui peut être activé ou désactivé, représentant un état binaire tel que "activé" ou "désactivé".

- Hérite de la classe CompoundButton

<code>android:textOn</code>	Label pour l'état activé
<code>android:textOff</code>	Label pour l'état non activé

```
<ToggleButton    android:layout_width="264dp"
    android:layout_height="wrap_content"
    android:id="@+id/toggleButtonId"
    android:textOn="Button ON"
    android:textOff="Button OFF"
    android:checked="true"
    android:layout_gravity= "center">
</ToggleButton>
```



25

A.Hammache

## 12/Images(ImageView et ImageButton):

### ImageView :

- Utilisé pour afficher des images statiques dans une application Android.
- Ne réagit pas aux événements de clic par défaut.
- Idéal pour afficher des illustrations, des photos ou des icônes non interactives.

### ImageButton :

- Utilisé pour créer des boutons avec des images au lieu de texte.
- Réagit aux événements de clic comme un bouton standard.
- Peut être personnalisé avec différentes images pour représenter différents états (normal, enfoncé, désactivé, etc.).

## Images (ImageView et ImageButton)

- Héritent de la classe View. Permet d'insérer des images
- Sont analogues à TextView et Button respectivement

`android:src` ; spécifie l'image utilisée (de type drawable généralement); on peut utiliser `setImageURI()`.  
l'attribut `android:background`

```
<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:orientation="vertical" >  
        <ImageView  
            android:id="@+id/myImageView1"  
            android:layout_width="313dp"  
            android:layout_height="349dp"  
            android:background="@drawable/umtoto" />  
        <ImageButton  
            android:id="@+id/monImageBtn1"  
            android:layout_width="254dp"  
            android:layout_height="131px"  
            android:background="@drawable/inscription" />  
    </LinearLayout>
```



## 13/La gestion des événements:

- Une interface homme-machine (IHM) répond aux actions de l'utilisateur, comme les clics, les sélections, etc.
- Chaque action génère un événement.
- Deux méthodes principales pour gérer ces événements :
  - Utilisation d'XML : Définit des comportements d'événements directement dans le code XML des éléments de l'IHM.

- Utilisation des écouteurs d'événements en Java : Méthode recommandée, où des classes Java sont attachées aux éléments de l'IHM pour détecter et répondre aux événements de manière précise.

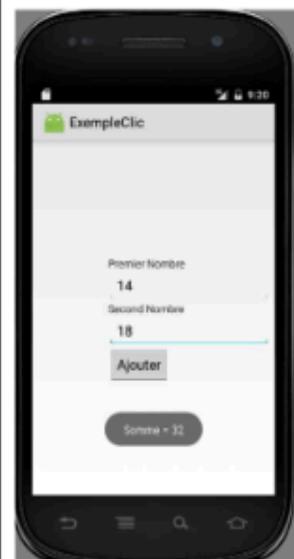
Le processus général de gestion des événements dans le développement d'applications :

1. Une action se produit sur un composant source
2. Le composant source génère un objet de type événement
3. La source transmet l'événement à son écouteur
4. L'écouteur appelle la méthode correspondante au type d'événement
5. La méthode en question spécifie les traitements à réaliser lorsqu'un événement du type correspondant se produit

### Exemple :

```
public class MainActivity extends Activity implements View.OnClickListener {
    EditText premierNombre;
    EditText secondNombre;
    Button btnAjout;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        premierNombre = (EditText) findViewById(R.id.premierNom);
        secondNombre = (EditText) findViewById(R.id.secondNom);
        btnAjout = (Button) findViewById(R.id.ajouterBtn);
        btnAjout.setOnClickListener(this);
    }
    @Override
    public void onClick(View view) {
        if(premierNombre.getText().toString().isEmpty() || secondNombre.getText().toString().isEmpty())
        {
            Toast.makeText(getApplicationContext(), "Remplir SVP les Champs",
            Toast.LENGTH_SHORT).show();
        }
        else
        {
            int num1 = Integer.parseInt(premierNombre.getText().toString());
            int num2 = Integer.parseInt(secondNombre.getText().toString());
            Toast.makeText(getApplicationContext(), "Somme = " + (num1 + num2),
            Toast.LENGTH_SHORT).show();
        }
    }
}
```



Ce code représente une activité Android (`MainActivity`) qui implémente `View.OnClickListener`. Il capture le clic sur un bouton (`btnAjout`) et effectue une action en conséquence. Voici une explication ligne par ligne :

1. `public class MainActivity extends Activity implements View.OnClickListener {` : Définition de la classe MainActivity qui étend Activity et implémente l'interface View.OnClickListener pour gérer les clics sur les vues.
2. `EditText premierNombre;` et `EditText secondNombre;` : Déclarations des objets EditText qui seront utilisés pour saisir les nombres.

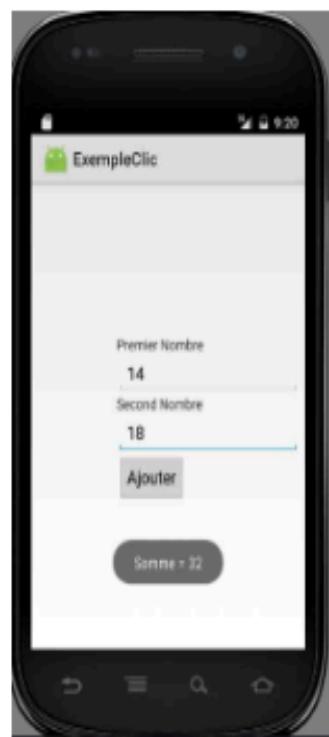
3. `Button btnAjout;` : Déclaration de l'objet Button qui sera utilisé pour déclencher l'ajout des nombres.
4. `@Override` : Annotation indiquant que la méthode qui suit redéfinit une méthode de la classe parente.
5. `protected void onCreate(Bundle savedInstanceState) {` : Début de la méthode onCreate(), qui est appelée lorsque l'activité est créée.
6. `super.onCreate(savedInstanceState);` : Appel de la méthode onCreate() de la classe parente (Activity) pour effectuer les initialisations de base.
7. `setContentView(R.layout.activity_main);` : Définition de la vue de l'activité en utilisant le fichier XML `activity_main.xml`.
8. `premierNombre = (EditText) findViewById(R.id.premierNom); et secondNombre = (EditText) findViewById(R.id.secondNom);` : Récupération des références des EditText dans le layout en utilisant leurs IDs.
9. `btnAjout = (Button) findViewById(R.id.ajouterBtn);` : Récupération de la référence du bouton dans le layout en utilisant son ID.
10. `btnAjout.setOnClickListener(this);` : Définition de l'activité comme écouteur de clic pour le bouton `btnAjout`.
11. `@Override` : Annotation indiquant que la méthode qui suit redéfinit une méthode de l'interface parente (View.OnClickListener).
12. `public void onClick(View view) {` : Début de la méthode onClick(), qui est appelée lorsque le bouton est cliqué.
13. `if(premierNombre.getText().toString().isEmpty() || secondNombre.getText().toString().isEmpty())` : Vérifie si l'un des champs de saisie est vide.
14. `{` : Début du bloc d'instructions exécuté si l'une des conditions précédentes est vraie.
15. `Toast.makeText(getApplicationContext(), "Remplir SVP les Champs", Toast.LENGTH_SHORT).show();` : Affiche un message Toast indiquant à l'utilisateur de remplir les champs.
16. `}` : Fin du bloc d'instructions exécuté si l'une des conditions précédentes est vraie.
17. `else {` : Début du bloc d'instructions exécuté si aucune des conditions précédentes n'est vraie.
18. `int num1 = Integer.parseInt(premierNombre.getText().toString()); et int num2 = Integer.parseInt(secondNombre.getText().toString());` : Convertit le contenu des EditText en entiers.
19. `Toast.makeText(getApplicationContext(), "Somme = " + (num1 + num2), Toast.LENGTH_SHORT).show();` : Affiche un message Toast avec la somme des deux nombres saisis.
20. `}` : Fin du bloc d'instructions exécuté si aucune des conditions précédentes n'est vraie.

En résumé, ce code crée une activité Android qui affiche deux champs de saisie et un bouton. Lorsque le bouton est cliqué, il vérifie si les champs de saisie sont vides. Si c'est le

cas, il affiche un message demandant à l'utilisateur de remplir les champs. Sinon, il additionne les nombres saisis et affiche le résultat dans un message Toast.

### Exemple :

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final EditText premierNombre = (EditText) findViewById(R.id.premierNom);
        final EditText secondNombre = (EditText) findViewById(R.id.secondNom);
        Button btnAjout = (Button) findViewById(R.id.ajouterBtn);
        btnAjout.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(premierNombre.getText().toString().isEmpty() ||
                    secondNombre.getText().toString().isEmpty())
                {
                    Toast.makeText(getApplicationContext(), "Remplir SVP les Champs",
                    Toast.LENGTH_SHORT).show();
                }
                else {
                    int num1 = Integer.parseInt(premierNombre.getText().toString());
                    int num2 = Integer.parseInt(secondNombre.getText().toString());
                    Toast.makeText(getApplicationContext(), "Somme = " + (num1 + num2),
                    Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}
```



1. Déclaration de la classe `MainActivity`, qui hérite de la classe `Activity`, indiquant qu'il s'agit d'une activité Android.
2. Redéfinition de la méthode `onCreate(Bundle savedInstanceState)` qui est appelée lors de la création de l'activité.
3. Appel de la méthode `onCreate()` de la classe parente (`super.onCreate(savedInstanceState)`), qui effectue les initialisations de base de l'activité.
4. Définition de la vue de l'activité en utilisant le layout défini dans `activity_main.xml` via `setContentView(R.layout.activity_main)`.
5. Récupération des références des `EditText` (`premierNombre` et `secondNombre`) et du bouton (`btnAjout`) en utilisant leurs IDs respectifs.
6. Définition d'un écouteur de clic anonyme pour le bouton en utilisant `btnAjout.setOnClickListener(...)`. Ce nouvel écouteur de clic est défini directement à l'intérieur de cette méthode.

7. Définition de la méthode `onClick(View v)` pour traiter les clics sur le bouton.

Cette méthode vérifie d'abord si l'un des champs de saisie est vide. Si c'est le cas, elle affiche un message Toast demandant à l'utilisateur de remplir les champs. Sinon, elle convertit le contenu des EditText en entiers, calcule la somme et affiche le résultat dans un message Toast.

Ce code suit le même principe que le précédent, mais avec une organisation différente de l'écouteur de clic pour le bouton. Au lieu d'implémenter `View.OnClickListener` dans la classe `MainActivity`, il utilise une implémentation anonyme directement dans la méthode `setOnClickListener()`.

Catégorie	Interface	Méthode
	<code>OnClickListener</code>	<code>onClick()</code>
	<code>OnLongClickListener</code>	<code>onLongClick()</code>
	<code>OnFocusChangeListener</code>	<code>onFocusChange()</code>
	<code>OnKeyListener</code>	<code>onKey()</code>
	<code>OnCheckedChangeListener</code>	<code>onCheckedChanged()</code>
	<code>OnItemSelectedListener</code>	<code>onItemSelected()</code>
	<code>OnTouchListener</code>	<code>onTouch()</code>

## Chapitre 5: les intents

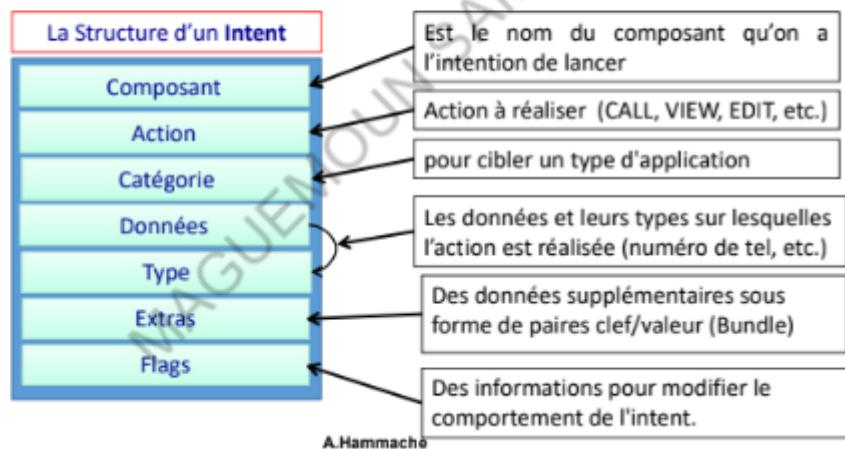
### 1/Définition d'un intent:

L'Intent, pilier du développement Android, facilite la communication entre les composants intra ou inter-applications en permettant le transfert de données. Ses types explicite et implicite permettent respectivement une communication directe entre des composants connus et la délégation de l'action à réaliser au système. La classe Intent, située dans le package android.content, offre les outils essentiels pour configurer, envoyer, et manipuler les données échangées entre les composants.

### 2/Description d'un intent:

Nous pouvons considérer l'objet intent comme un “**message**” contenant :

- Des informations nécessaires cas explicite (exemple : Composant)
- Des informations nécessaires cas implicite (exemple : action et données).



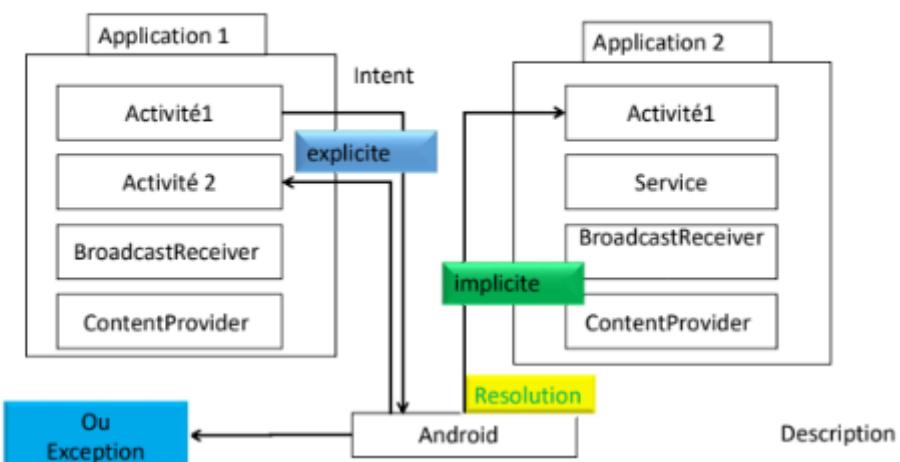
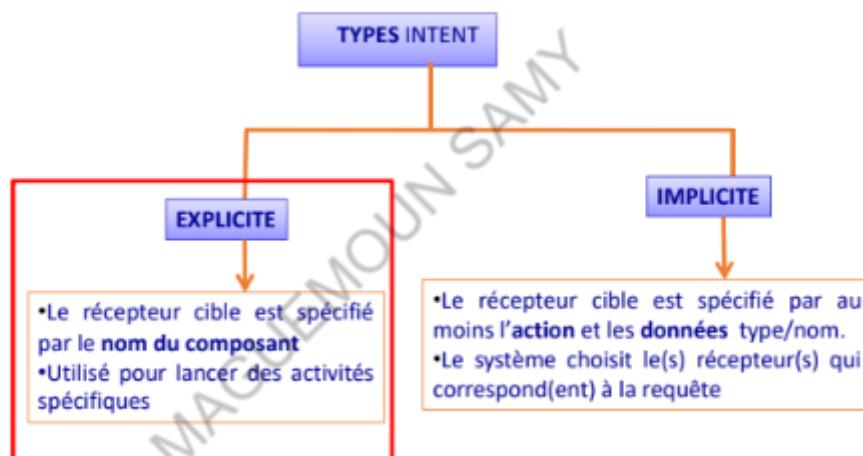
Voici une présentation simplifiée des composants d'un Intent en Android :

1. Composant : C'est le nom du composant que l'on a l'intention de lancer. Dans le cas d'un Intent explicite, il spécifie directement le composant de destination par son nom de classe.
2. Action : Il s'agit de l'action à réaliser, comme CALL pour effectuer un appel téléphonique, VIEW pour afficher quelque chose, ou EDIT pour modifier des données.
3. Données : Il représente les données et leurs types sur lesquelles l'action est réalisée, telles qu'un numéro de téléphone pour une action CALL ou une adresse URL pour une action VIEW.
4. Catégorie : Elle permet de cibler un type d'application spécifique en ajoutant des catégories supplémentaires à l'intent. Par exemple, une catégorie BROWSABLE indique que l'intention peut être gérée par un navigateur web.

5. Extras : Ce sont des données supplémentaires sous forme de paires clé-valeur, souvent utilisées pour transmettre des informations spécifiques entre les composants.
6. Flags : Ces informations permettent de modifier le comportement de l'intent, par exemple pour spécifier des options supplémentaires ou des indicateurs de traitement.
7. Type : Il indique le type de données associées à l'intent, tel qu'un type MIME pour spécifier le type de fichier ou de données à manipuler.

En utilisant ces composants, un Intent peut être configuré pour effectuer une action spécifique, transmettre des données, cibler un composant particulier, et modifier son comportement selon les besoins de l'application Android.

### 3/Types d'intent :



## Intent Explicite:

Dans l'Intent explicite, le composant récepteur, dans ce cas une activité à lancer, est spécifié par le nom de la classe du composant. La création d'un tel Intent se fait comme suit :

- **Context** : Il s'agit du contexte à partir duquel l'Intent est créé. En général, cela fait référence à l'activité en cours, souvent passé en tant que `this`.
- **Cls** : Il s'agit de la classe Java héritant de la classe `Activity` que vous souhaitez lancer.
- **startActivity()** : Cette méthode est utilisée pour lancer l'activité spécifiée dans la classe `Cls`. Il ne s'attend pas à un résultat de l'activité lancée.

Voici un exemple concret :

```
Intent i = new Intent(Context context, Class<?> cls);  
startActivity(i);
```

Dans cet exemple, `this` est passé comme contexte (faisant référence à l'activité en cours), et `AutreActivité.class` est la classe de l'activité que nous voulons lancer.

Donc, en résumé, l'Intent explicite est utilisé lorsque vous connaissez précisément l'activité que vous voulez démarrer, et vous la spécifiez en utilisant son nom de classe.

la différence réside dans la manière dont le contexte est spécifié :

1. Dans la première ligne :

```
Intent i = new Intent(Context context, Class<?> cls);
```

- Le contexte est spécifié de manière générale en tant que paramètre `context`, qui peut être n'importe quel objet de type `Context`.
- Cela signifie que vous pourriez théoriquement appeler cette ligne de code à partir de n'importe quel endroit où vous avez accès à un objet de type `Context`, tel qu'une activité, un service, etc.

2. Dans la deuxième ligne :

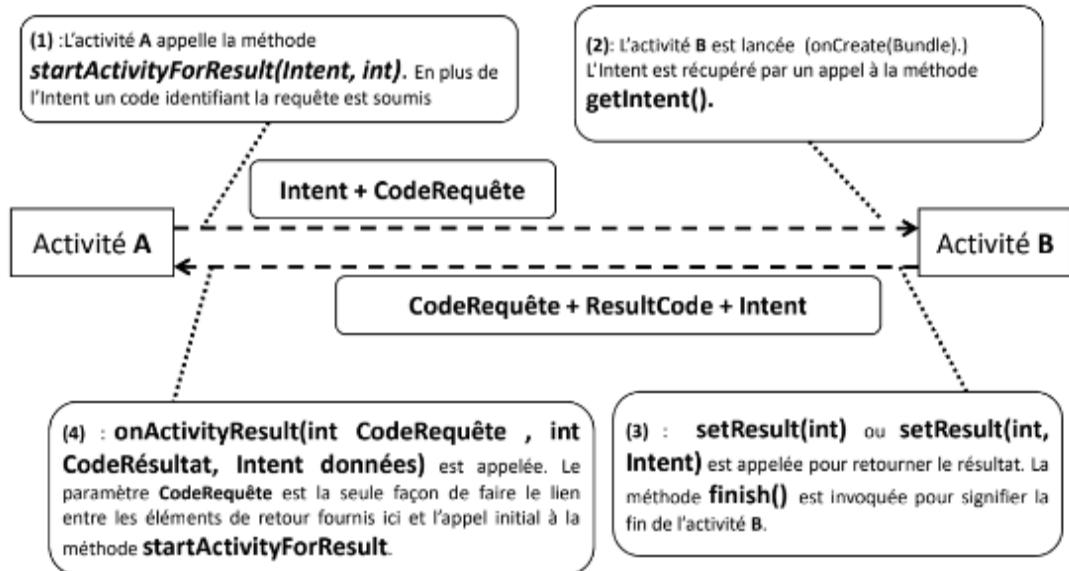
```
Intent i = new Intent(this, AutreActivité.class);
```

- Le contexte est spécifié de manière spécifique en tant que `this`, qui fait référence à l'activité actuelle à partir de laquelle l'Intent est créé.
- Cela signifie que cette ligne de code est généralement utilisée à l'intérieur d'une activité pour démarrer une autre activité dans la même application.

En résumé, la principale différence réside dans la spécificité du contexte. La première méthode est plus générale et peut être utilisée dans différents contextes, tandis que la deuxième méthode est plus spécifique à l'intérieur d'une activité pour lancer une autre activité dans la même application.

- Intent explicite (avec résultats):

On peut aussi lancer une activité, qui retourne des résultats:



L'activité A va lancer une intent avec un code identifiant la requête avec `startActivityForResult(intent,int)`, le résultat va être envoyé vers l'activité B ensuite avec `getIntent()` l'intent envoyé par A sera capté par B, après l'activité B va construire une autre intent avec `setResult(int)` qui va retourner le résultat à l'activité A.

<b>PremièreActivité</b>	
<pre> public final static int CODE= 0; Intent intent = new Intent(this, SecondeActivité.class); startActivityForResult(intent, CODE); // Appelée lorsque la Seconde Activité termine son travail public void onActivityResult(int CodeRequete, int CodeRésultat, Intent intent_resultat) {     if (CodeRequete ==CODE)         if(CodeRésultat==RESULT_OK)             int res= intent_resultat.getIntExtra ("RESULTAT",0);         ..... } </pre>	1

<b>SecondeActivité</b>	
<pre> Intent reçu= getIntent(); Intent envoi=new Intent(); envoi.putExtra("RESULTAT", 5); setResult(RESULT_OK, envoi); finish(); </pre>	2
	3

Ce code illustre le processus d'utilisation de `startActivityForResult()` pour démarrer une activité (seconde activité) depuis une première activité, puis récupérer des données de la seconde activité lorsque celle-ci termine son travail.

Dans la première activité :

- Déclaration du code de requête** : La constante `CODE` est définie pour identifier cette requête. C'est une bonne pratique d'utiliser une constante pour identifier les requêtes afin de rendre le code plus lisible et maintenable.
- Création et démarrage de l'intent** : Un Intent est créé pour démarrer la seconde activité (`SecondeActivité.class`) en utilisant `startActivityForResult()`, qui permet de démarrer une activité et d'attendre un résultat de retour.
- Méthode `onActivityResult()`** : Cette méthode est appelée lorsque la seconde activité termine son travail et renvoie un résultat à la première activité. Le paramètre `CodeRequete` correspond au code de requête utilisé pour démarrer l'activité. `CodeRésultat` indique le résultat du traitement (par exemple, `RESULT_OK` pour indiquer que tout s'est bien passé). `intent_resultat` contient les données renvoyées par la seconde activité.

Dans la seconde activité :

- Récupération de l'intent reçu** : L'intent reçu de la première activité est récupéré à l'aide de `getIntent()`.
- Préparation de l'intent de retour** : Un nouvel Intent (`envoi`) est créé pour contenir les données à retourner à la première activité. Dans cet exemple, un entier est ajouté en extra avec la clé "`RESULTAT`".
- Définition du résultat et envoi** :  `setResult(RESULT_OK, envoi)` est utilisé pour spécifier le résultat de l'activité (`RESULT_OK` indique que tout s'est bien passé) et l'intent contenant les données à retourner. Enfin, `finish()` est appelé pour terminer l'activité et renvoyer le résultat à l'activité appelante.

Ainsi, dans la méthode `onActivityResult()` de la première activité, vous pouvez récupérer les données renvoyées par la seconde activité et effectuer des traitements supplémentaires en fonction de ces données.

### Intent implicite:

Dans un Intent implicite, le composant récepteur n'est pas spécifié directement par le programmeur. C'est le système Android qui détermine le ou les composants pouvant répondre à cet Intent en se basant sur des critères tels que le nom de l'action, les données associées (URI), la catégorie, etc.

- Actions** : Les actions peuvent être prédéfinies (natives) dans Android, comme `Intent.ACTION_CALL` qui est utilisée pour démarrer un appel téléphonique. Elles peuvent également être définies par le programmeur pour des besoins spécifiques.
- Résolution de l'Intent** : La résolution d'un Intent se fait en comparant l'objet Intent avec le `IntentFilter` de chaque composant installé sur le dispositif. Si un composant correspond aux critères spécifiés dans l'Intent, il est alors considéré comme apte à gérer cet Intent.

Voici un exemple concret :

```

Uri numero = Uri.parse("tel:0555555555");

Intent appel = new Intent(Intent.ACTION_CALL, numero);

startActivity(appel);

```

Dans cet exemple, un Intent implicite est créé pour effectuer un appel téléphonique vers le numéro spécifié. L'action `Intent.ACTION_CALL` indique à Android qu'il s'agit d'une demande de démarrage d'appel téléphonique. La donnée associée (`numero`) est spécifiée sous forme d'URI, indiquant le numéro de téléphone à appeler. Lorsque `startActivity(appel)` est appelé, Android résoudra cet Intent et démarrera l'application de téléphone par défaut pour effectuer l'appel.

#### ■Quelques actions prédéfinies :



Nom d'Action	Description
<code>ACTION_ANSWER</code>	Prendre en charge un appel entrant.
<code>ACTION_CALL</code>	Appeler un numéro de téléphone.
<code>ACTION_SENDTO</code>	Envoyer des données texte ou binaires par SMS.
<code>ACTION_EDIT</code>	Editer des données
<code>ACTION_MAIN</code>	Démarrer comme un point d'entrée principale.
<code>ACTION_PICK</code>	Prendre un élément à partir d'une source de données.
<code>ACTION_VIEW</code>	Afficher des données pour l'utilisateur.
<code>ACTION_SEARCH</code>	Démarrer une activité de recherche.

Voici un résumé du code qui utilise un Intent implicite pour afficher les contacts sur un appareil Android

#### Code java:

```

String données = "content://com.android.contacts/contacts/"; 1

Uri uri = Uri.parse(données); 2

Intent affiche = new Intent(Intent.ACTION_VIEW, uri); 3

startActivity(affiche); 4

```

### **Explication du code:**

**1/Définir l'URI des contacts** : Cette chaîne représente l'URI de la base de données des contacts.

**2/Convertir la chaîne en URI** : La chaîne est convertie en objet URI.

**3/Créer un Intent implicite :**

- L'Intent est créé avec l'action `Intent.ACTION_VIEW` et l'URI des contacts, indiquant qu'il doit afficher les contacts.

**4/Lancer l'activité :**

- L'Intent est envoyé au système Android, qui choisit l'application appropriée pour afficher les contacts.

### **Résumé**

Le code crée un Intent implicite pour afficher les contacts en utilisant `Intent.ACTION_VIEW` avec l'URI des contacts et lance l'activité correspondante.

Voici un autre exemple d'utilisation d'un Intent implicite pour effectuer une recherche sur le web dans une application Android :

### **Code java:**

```
Intent recherche = new Intent(Intent.ACTION_WEB_SEARCH); 1  
  
String requête = "applications Android"; 2  
  
recherche.putExtra(SearchManager.QUERY, requête); 2  
  
if (recherche.resolveActivity(getApplicationContext()) != null) { 3  
  
    startActivity(recherche); 3  
  
}
```

### **Explication du code:**

**1/Créer un Intent implicite pour la recherche web :**

- L'Intent est créé avec l'action `Intent.ACTION_WEB_SEARCH`, indiquant que l'Intent est destiné à effectuer une recherche sur le web.

## 2/ Définir la requête de recherche :

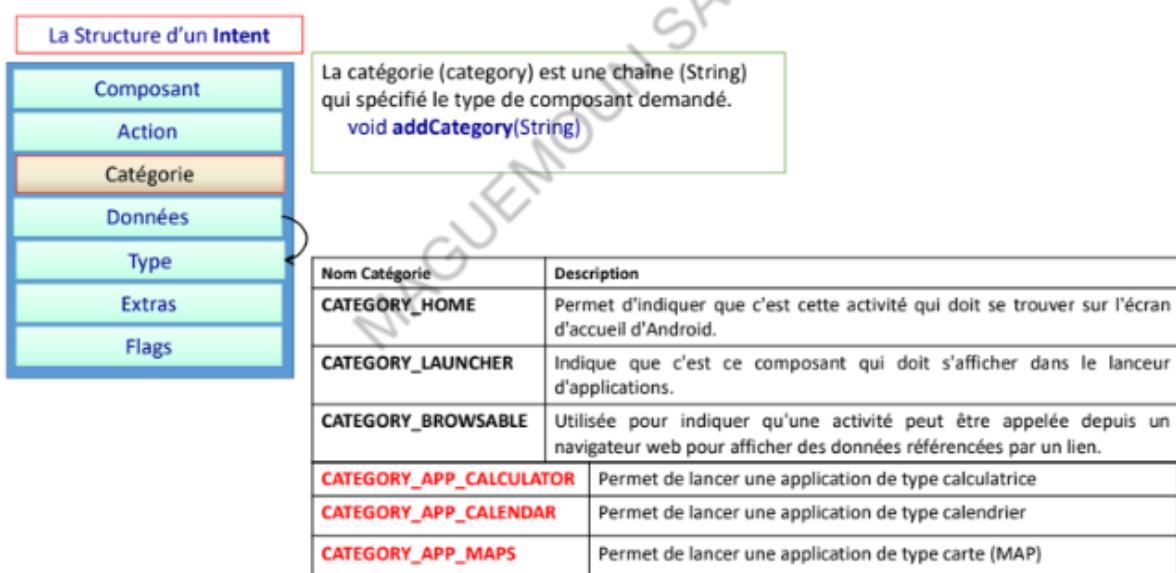
- Une chaîne de caractères contenant la requête de recherche est définie.
- La méthode `putExtra` ajoute la requête de recherche à l'Intent en tant qu'extra, avec la clé `SearchManager .QUERY`.

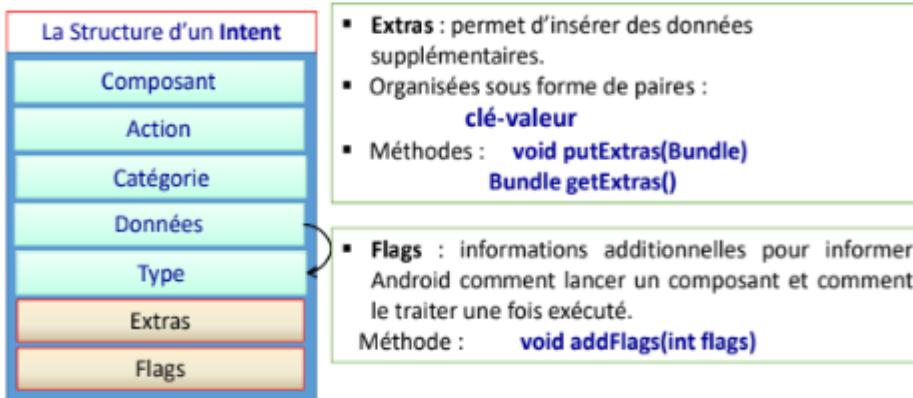
## 3/Vérifier si une activité peut gérer l'Intent :

- La méthode `resolveActivity` utilise le `PackageManager` pour vérifier s'il existe une activité capable de gérer l'Intent.
- Si une activité correspondante est trouvée (`resolveActivity` ne renvoie pas `null`), l'Intent est lancé avec `startActivity`.

## Résumé

Le code crée un Intent implicite pour effectuer une recherche web avec une requête spécifiée. Il vérifie d'abord si une application capable de gérer cette recherche est disponible, puis lance cette application pour afficher les résultats de la recherche.





Nom Flags	Description
<code>FLAG_ACTIVITY_NEW_TASK</code>	Permet de lancer une activité dans une nouvelle tâche, sauf si cette dernière existe déjà dans une tâche.
<code>FLAG_ACTIVITY_SINGLE_TOP</code>	Permet de lancer une nouvelle instance de l'activité si elle n'est pas au top de la pile.
<code>FLAG_ACTIVITY_CLEAR_TOP</code>	Permet de fermer toutes les activités qui se trouvent au-dessus d'elle.

## 4/Résolution d'un intent:

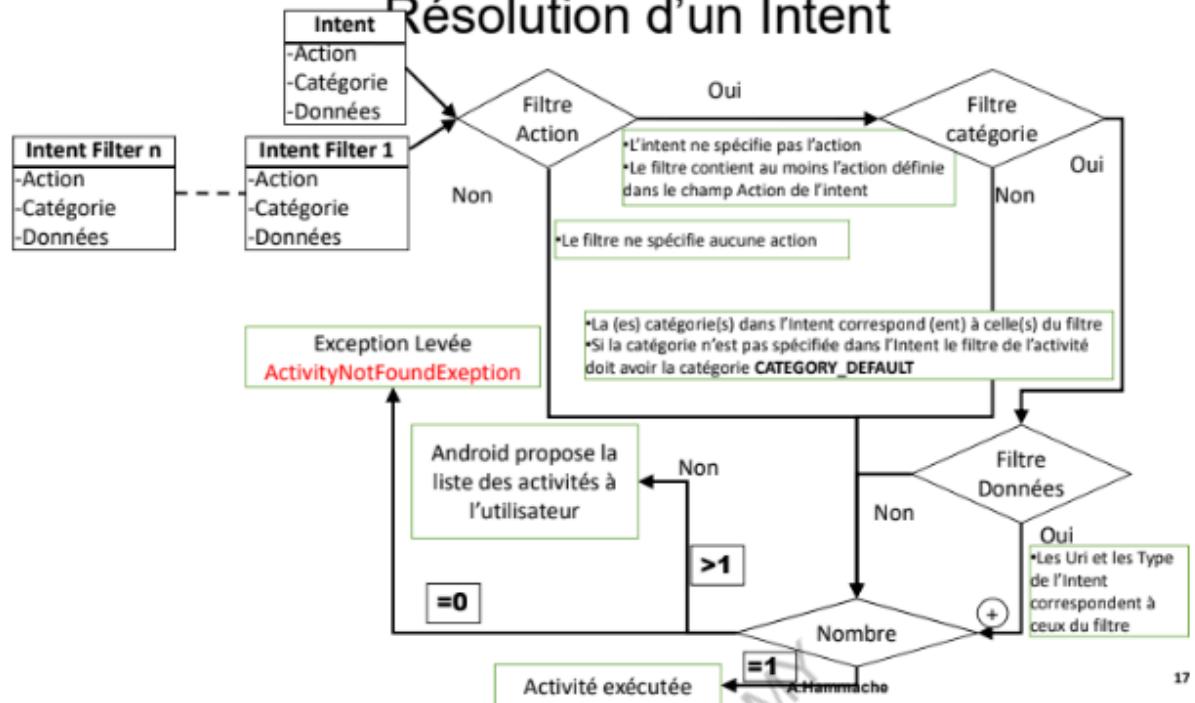
La résolution des intents dans Android se fait en comparant l'objet Intent avec les IntentFilter des composants. Un IntentFilter est attaché à un composant (comme une activité) dans le fichier manifeste.

Voici un exemple de configuration d'une activité avec un IntentFilter dans le fichier manifest :

```
<activity
    android:name=".IntentTesterActivity"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <action android:name="android.intent.action.VIEW"/>
        <action android:name="android.intent.action.DELETE"/>
        <category android:name="android.intent.category.LAUNCHER" />
        <data android:mimeType="video/mpeg" />
    </intent-filter>
</activity>
```

Dans cet exemple, l'activité `IntentTesterActivity` peut répondre à plusieurs actions (`MAIN`, `VIEW`, `DELETE`), appartient à la catégorie `LAUNCHER`, et peut traiter des données de type MIME `video/mpeg`.

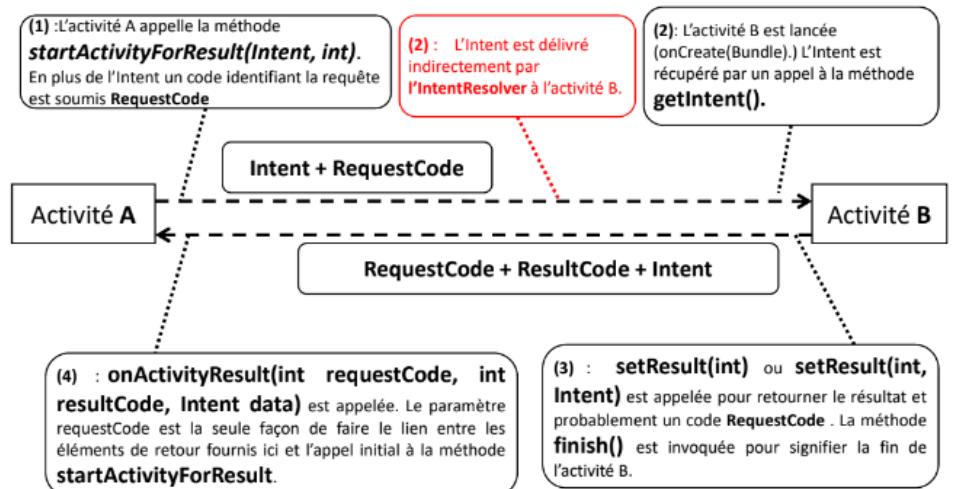
## Résolution d'un Intent



L'intent resolver se charge de trouver le composant approprié pour exécuter l'activité, étant donné que dans les intents implicite le nom du composant n'est pas spécifié c'est le système (plus précisément l'intent resolver) qui se charge de chercher le composant approprié en dépend de l'action, la catégorie et les données.

### **3/L'intent implicite (avec résultats) :**

- On peut aussi lancer une activité, qui retourne des résultats:



## Résumé des étapes de la résolution d'un intent Android

### 1. Création d'un intent:

- L'application crée un objet `Intent` en spécifiant l'action à effectuer, les données à transmettre et les composants cibles (optionnel).

### 2. Envoi de l'intent:

- L'application appelle la méthode `startActivity()`, `startService()` ou `sendBroadcast()` pour envoyer l'intent au système Android.

### 3. Résolution de l'intent:

- Le système Android utilise l'`IntentResolver` pour trouver l'application la mieux adaptée pour gérer l'intent.
  - `IntentResolver` compare l'action, les données et les catégories de l'intent avec les filtres d'intent des applications installées.
  - Le filtre d'intent avec la priorité la plus élevée et la meilleure correspondance est sélectionné.

### 4. Lancement du composant:

- Si un seul composant correspond à l'intent, il est lancé.
  - L'intent est passé au composant lancé via la méthode `onCreate()` ou `onBind()`.
- Si plusieurs composants correspondent à l'intent, l'utilisateur peut choisir l'application à utiliser via un sélecteur d'intent.

### 5. Gestion de l'intent:

- Le composant lancé traite l'intent en fonction de son type (activité, service, broadcast).
  - L'intent peut être utilisé pour récupérer des données, démarrer d'autres activités, interagir avec des ressources système, etc.

### 6. Retour des résultats (optionnel):

- Une activité peut renvoyer des résultats à l'application appelante via la méthode `setResult()`.
  - Les résultats peuvent inclure des données et un code de statut indiquant si l'opération a réussi.

### 7. Fin du composant:

- Le composant appelle la méthode `finish()` pour se terminer et libérer les ressources.