

# Introduction

Systèmes d'exploitation (OS : Operating System) pour les terminaux mobiles :

- Pour les smartphones et les tablettes
- Android (Google), iOS (Apple), Windows mobile, Blackberry, etc.

Android :

- Un OS basé sur Linux, pour les smartphones et les tablettes.
- Un OS open source
- Développé par Google
- Initialement conçu pour les smartphones et tablettes tactiles, puis s'est diversifié : les télévisions (Android TV), les voitures (Android Auto), les ordinateurs (Android-x86) et les smartwatch (Android Wear).
- Le système d'exploitation mobile le plus utilisé au monde

Parts de marché des OS mobiles en 2020 : ([www.idc.com/promo/smartphone-market-share/os](http://www.idc.com/promo/smartphone-market-share/os))

- Android 85%
- iOS : 15%
- Autres : 0%

Développement des applications Android :

- Développement avec Java sous « **Android Studio** » : génération du fichier exécutable « **.apk** »
- Développement avec JavaScript et Cordova : conversion en Java et génération du fichier exécutable « **.apk** »
- L'exécution de l'exécutable « **.apk** » nécessite la machine virtuelle Dalvik (pour les anciennes versions du OS) ou ART (pour les nouvelles versions du OS).

Télécharger une application Android :

- Google Play
- Market alternatifs (Amazon Appstore for Android, GetJar, F-Droid, Samsung Galaxy Apps, etc.)
- Téléchargement direct du fichier binaire (.apk) sur le smartphone

Google Play :

- Un magasin d'applications créé par Google (le 6 mars 2012)
- Une boutique d'applications pour le système d'exploitation Android
- En 2020, Google Play contient 3 000 000 applications, ce qui en fait le plus gros magasin d'applications au monde devant iOS et Windows Phone (<https://www.appbrain.com/stats>)

## Les Versions d'Android :

Version Android	Date de sortie	API Level	Nom
Android 11	?	30	Android 11
Android 10	3 Septembre 2019	29	Android 10
Android 9.0	1 Décembre 2018	28	Pie
Android 8.1	6 Décembre 2017	27	Oreo
Android 8.0	21 août 2017	26	Oreo
Android 7.1	4 Octobre 2016	25	Nougat
Android 7.0	Aout 2016	24	Nougat
Android 6.0	Aout 2015	23	Marshmallow
Android 5.1	Mars 2015	22	Lollipop
Android 5.0	Novembre 2014	21	Lollipop
Android 4.4W	Juin 2014	20	Kitkat Watch
Android 4.4	Octobre 2013	19	Kitkat
Android 4.3	Juillet 2013	18	Jelly Bean
Android 4.2	Novembre 2012	17	Jelly Bean
Android 4.1	Juin 2012	16	Jelly Bean
Android 4.0.3	Décembre 2011	15	Ice Cream Sandwich
Android 4.0	Octobre 2011	14	Ice Cream Sandwich
Android 3.2	Juin 2011	13	Honeycomb
Android 3.1	Mai 2011	12	Honeycomb
Android 3.0	Février 2011	11	Honeycomb
Android 2.3.3	Février 2011	10	Gingerbread
Android 2.3	Novembre 2010	9	Gingerbread
Android 2.2	Juin 2010	8	Froyo
Android 2.1	Janvier 2010	7	Eclair
Android 2.0.1	Décembre 2009	6	Eclair
Android 2.0	Novembre 2009	5	Eclair
Android 1.6	Septembre 2009	4	Donut
Android 1.5	Mai 2009	3	Cupcake
Android 1.1	Février 2009	2	Base
Android 1.0	Octobre 2008	1	Base

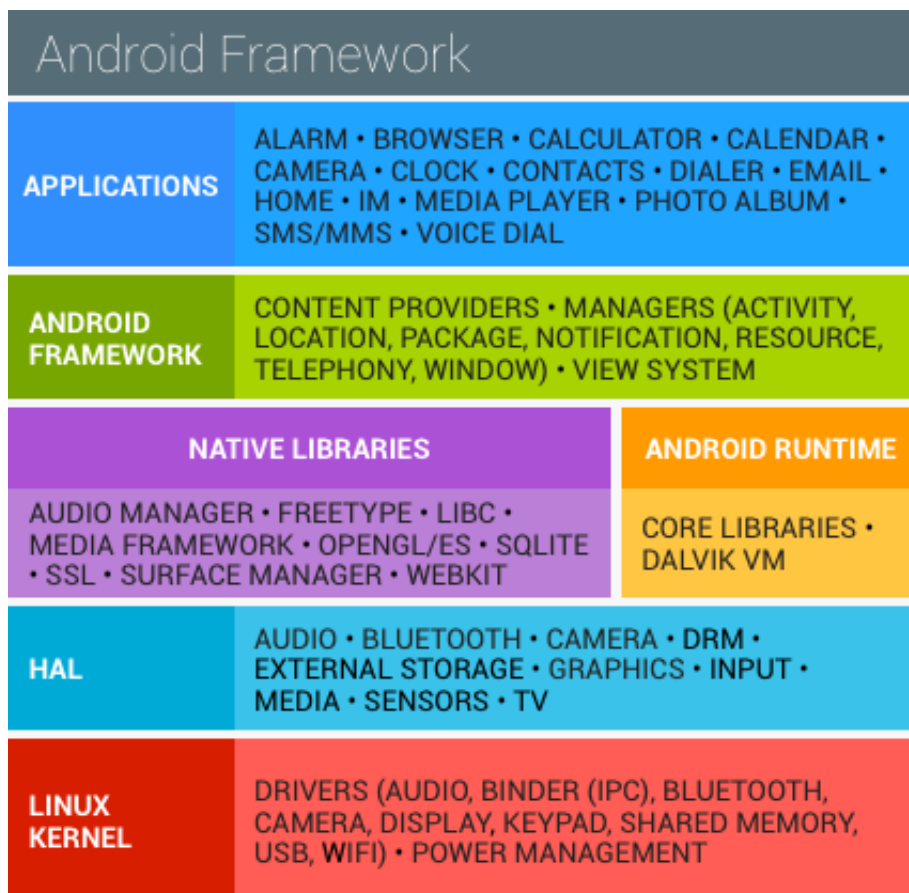
**Remarque 1 :** Chaque version d'Android est identifiée par un entier unique, nommé « API Level », et a un nom qui correspond à un dessert (sauf les versions  $\geq 10$ ) : Pie, Oreo, Nougat, Eclair, etc.

**Remarque 2 :** Une application développée avec une API (exemple l'API 21), fonctionne correctement sur les API les plus récentes (et les OS correspondant) mais ne peut pas fonctionner sur les API les plus anciennes.

Principales différences entre les versions :

- Correction de bugs
- Nouvelles APIs avec des nouvelles fonctionnalités
- Nouveau mode d'exécution : passage d'une VM (Dalvik) à une autre (ART)
- Optimisation énergétique
- Modification de l'apparence de l'UI (thèmes, notifications, fenêtrage, etc.)
- Amélioration de la sécurité
- Support des nouvelles technologies (5G, IoT, waterfall displays : écrans incurvés, etc.)

Pile logicielle Android :



## 1) Interface graphique

Activity : une activité est une fenêtre sous Android. Elle peut contenir des vues (View). Il y a deux types de vues :

- Les vues simples, telles que :
  - o Button : bouton
  - o TextView : étiquette
  - o EditText : champ de texte

- Les conteneurs de vues, tels que :
  - LinearLayout : un conteneur qui peut disposer les vues horizontalement ou verticalement
  - ListView : dispose les vues dans une liste

Une application peut contenir plusieurs activités. Il faut déclarer une activité par défaut qui s'affiche au lancement d'une activité.

## 2) Environnement de développement

Pour développer des applications Android, il faut :

- L'environnement de développement (IDE) « Android Studio »
- Le « Software Development Kit » (SDK) qui contient les bibliothèques (API) nécessaires pour développer des applications Android (c'est l'équivalent de la JDK dans le cas des applications bureautiques)
- Un smartphone Android pour tester les applications réalisées. Il y a 2 possibilités
  - On utilise un émulateur qui simule un Smartphone Android sur un ordinateur :
    - Si la machine supporte la Virtualisation, utiliser un émulateur **intel** (plus rapide)
    - Si la machine ne supporte pas la Virtualisation, utiliser un émulateur **arm** (lent)
    - Si la mémoire RAM de la machine est suffisante (> 4 Go), utiliser une version récente de l'émulateur
    - Si la mémoire RAM de la machine est limitée, utiliser une version ancienne de l'émulateur (exemple : API 19)
    - Il y a d'autres émulateurs à part l'émulateur officiel
  - On utilise un Smartphone Android réel connecté par câble USB à l'ordinateur : Il faut activer l'option « USB debugging » à partir de « paramètres > options développeur »

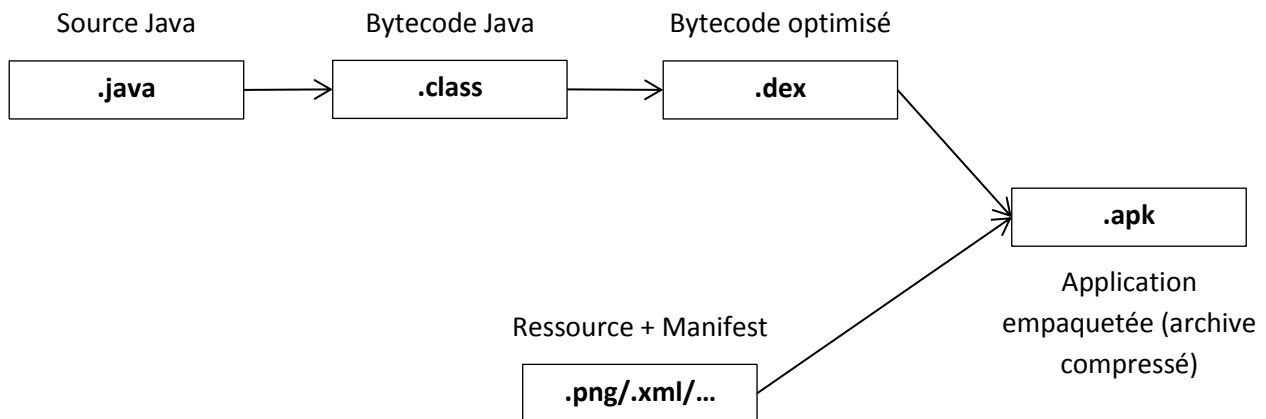
Nous allons installer « **Android Studio** » (dernière version 4.0.2) à partir de :  
<https://developer.android.com/studio/>

Android studio contient les 2 outils suivants :

- Software Development Kit (SDK) Manager : permet de visualiser les différentes versions de SDK (un numéro API correspond à chaque version de SDK) disponibles et les télécharger. Accès possible à partir de « **Tools > Android > SDK Manager** »
- Android Virtual Device (AVD) Manager : gère les différents émulateurs. Il est possible de créer plusieurs émulateurs pour les différentes tailles de tablettes et de Smartphones. Accès possible à partir de « **Tools > Android > AVD Manager** »

### 3) Compilation

Java permet d'obtenir un exécutable (.class, .jar, .war, .apk, etc.) qui tourne sur une machine virtuelle. La machine virtuelle pour les applications bureautiques et web est Java Virtual Machine (JVM) (le programme **java.exe** pour Windows). Pour Android il y a l'ancienne machine virtuelle Dalvik qui est remplacée par Android Runtime (ART).



### 4) Les outils « SDK Manager » et « AVD Manager »

Les outils « SDK Manager » et « AVD Manager » sont accessibles à partir de « Android Studio » ou à partir du dossier d'installation du SDK.

#### a) SDK Manager

Cet outil est accessible à partir de « Android Studio » : **Tools > Android > SDK Manager**

Il permet d'installer tous les éléments nécessaires pour programmer, tels que : API, Outils de compilation, Emulateurs, Etc.

#### b) AVD Manager

L'outil « AVD (Android Virtual Device) Manager » est accessible à partir de « Android Studio » :

**Tools > Android > AVD Manager**

Il permet de créer des nouveaux émulateurs avec différentes résolutions. Il permet aussi de lancer un émulateur.

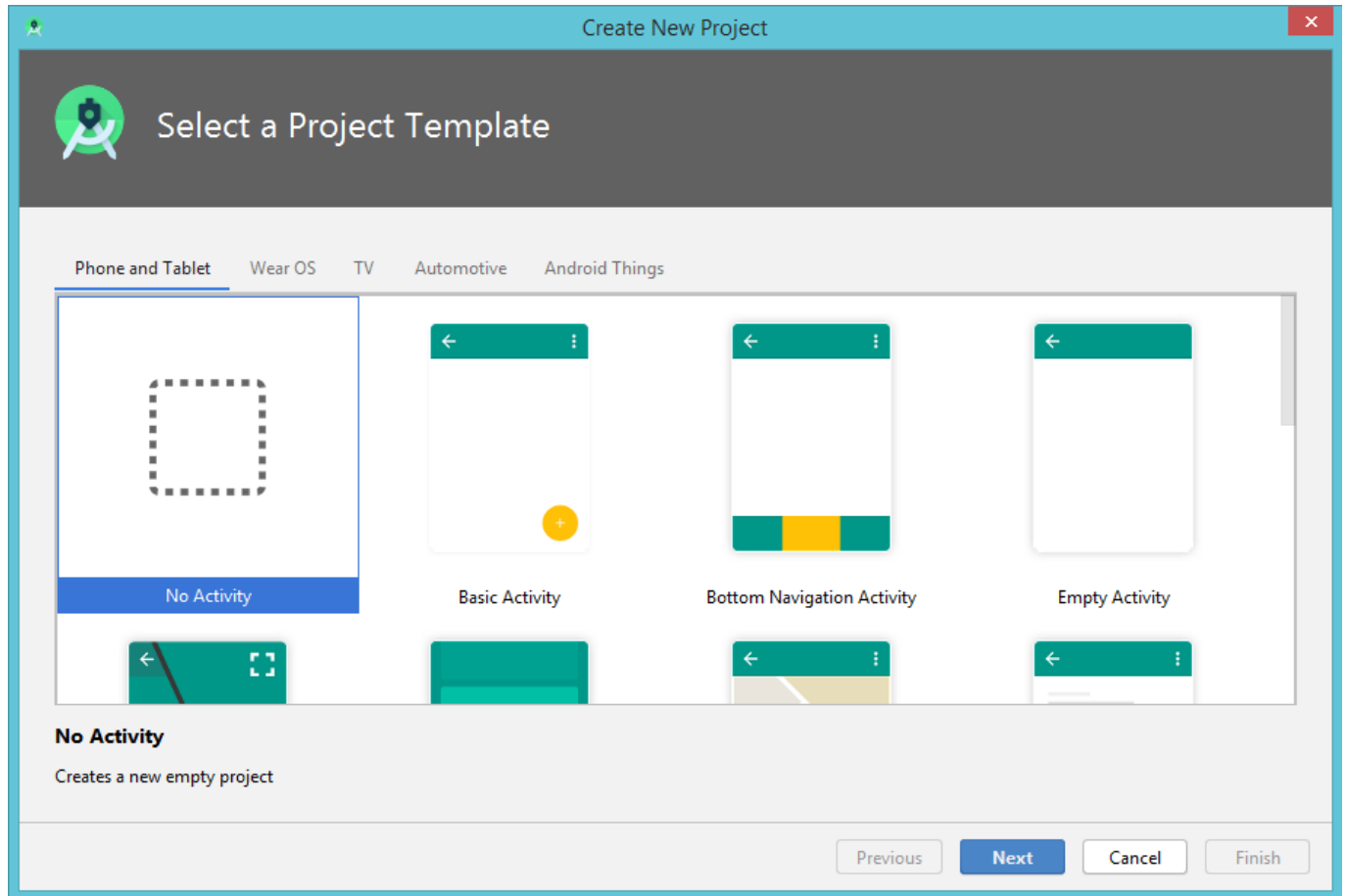
**Remarque :** Le lancement d'un émulateur peut être très lent (entre 1 à 10 min). Il est donc nécessaire de lancer l'émulateur une seule fois et de le garder ouvert pour l'utiliser avec tous les tests.

## 5) Projet Android

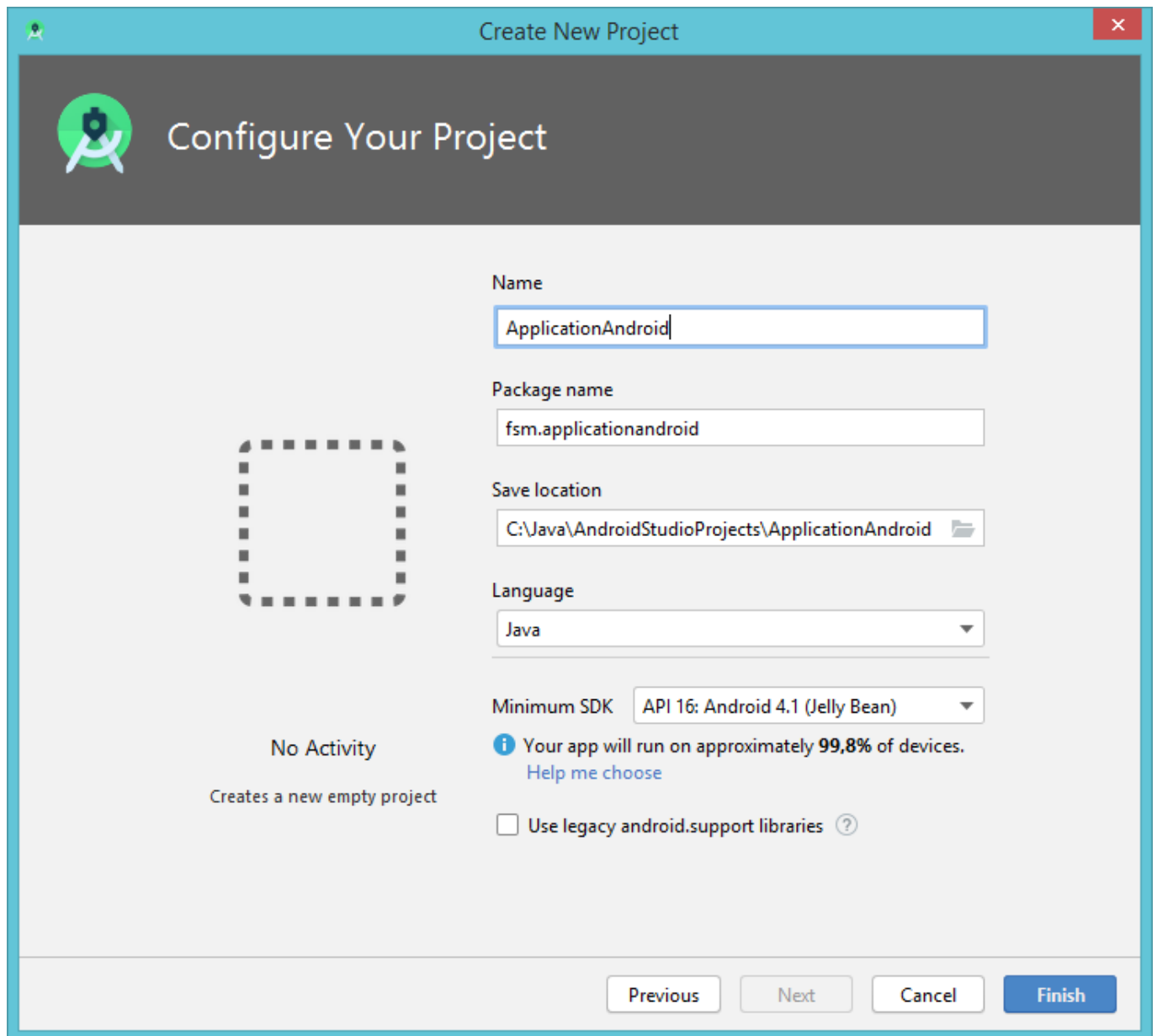
### a) Création d'un projet Android

Pour créer une nouvelle application Android, il faut créer un nouveau projet sous « Android Studio » en suivant les étapes suivantes :

- Clic sur « **File > New > New Project** » et la fenêtre « Create New Project » s'ouvre :



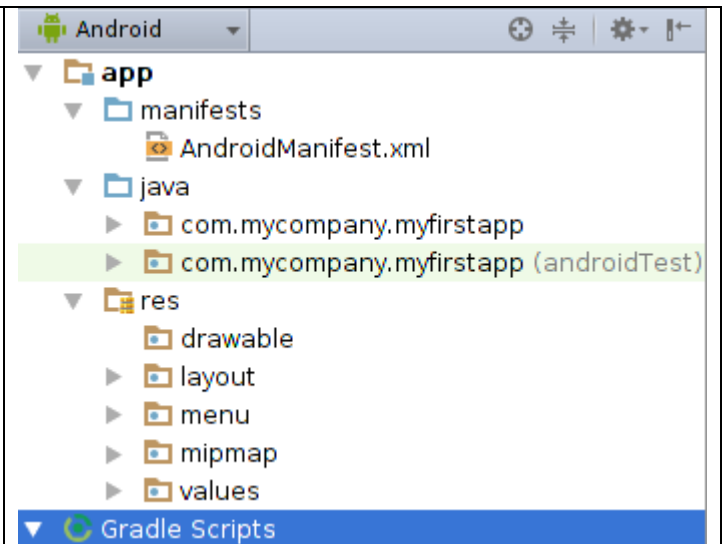
- Choisir « **No Activity** » et clic sur le bouton « **Next** »
- Dans la fenêtre qui s'ouvre, renseigner le nom de l'application dans le champ « **Name** », le nom du package et l'emplacement de sauvegarde du projet. Choisir le langage de programmation **Java** et Cliquer sur le bouton « **Finish** ».



## b) Structure d'un Projet Android

Chaque projet Android contient le module « **app** » qui regroupe l'ensemble des codes sources (fichiers java) et les fichiers ressources (fichiers layout, XML, images, constantes, etc.). Dans le module « app », les fichiers sont affichés dans les groupes suivants :

- **manifests** : contient le fichier « **AndroidManifest.xml** ». Ce fichier est essentiel et contient la liste des composants d'une application. Par exemple, il contient l'ensemble des activités disponibles et l'activité principale qui s'affiche au lancement de l'application.  
**Remarque** : si une activité n'est pas déclarée dans le fichier manifest, il n'est pas possible de l'afficher dans l'application.
- **Java** : contient tous les fichiers java
- **res** : contient tous les fichiers ressources (non java), tels que layouts, images, XML, etc.



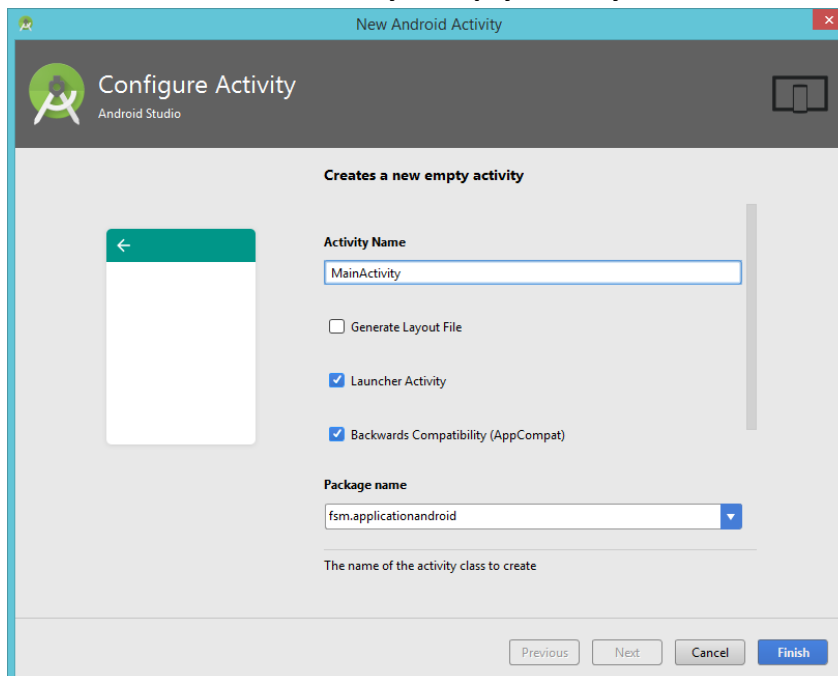
### c) Ajouter des fichiers au projet

Pour ajouter un fichier au projet, vérifier d'abord que le module « app » est sélectionné.

#### Ajouter une activité

Pour ajouter une activité au projet, il faut suivre les étapes suivantes :

- Clic sur « **File > New > Activity > Empty Activity** », et la fenêtre « **Configure Activity** » s'ouvre :



- Saisir un nom de cette activité dans le champ « **Activity Name** ». Décocher la case « **Generate Layout File** » Si cette activité doit s'afficher au moment de lancement de l'application, cocher « **Launcher Activity** », sinon décocher cette case. Clic sur le bouton « **Finish** » et une nouvelle activité s'ajoute au projet.

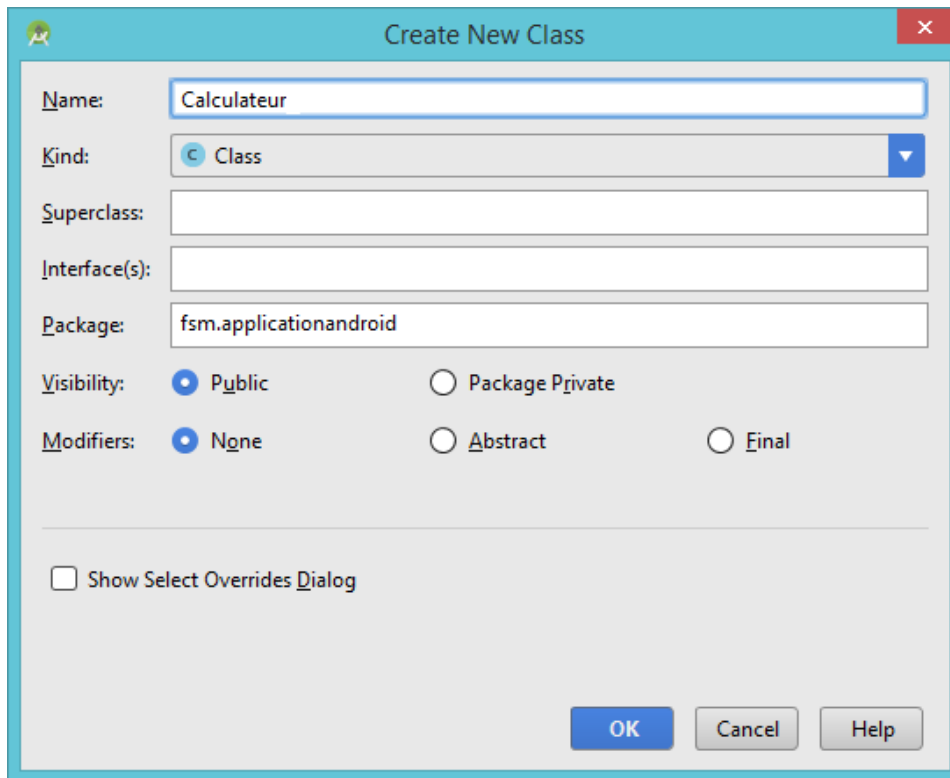


**Remarque** : Une activité est une classe Java qui doit être déclarée dans le fichier manifest du projet. Si on ajoute cette classe en suivant les étapes précédentes, « Android Studio » se charge de la déclarer dans le fichier manifest.

## Ajouter une classe Java

Pour ajouter une classe java ordinaire, il faut suivre les étapes suivantes :

- Clic sur « **File > New > Java Class** » et la fenêtre suivante s'ouvre :

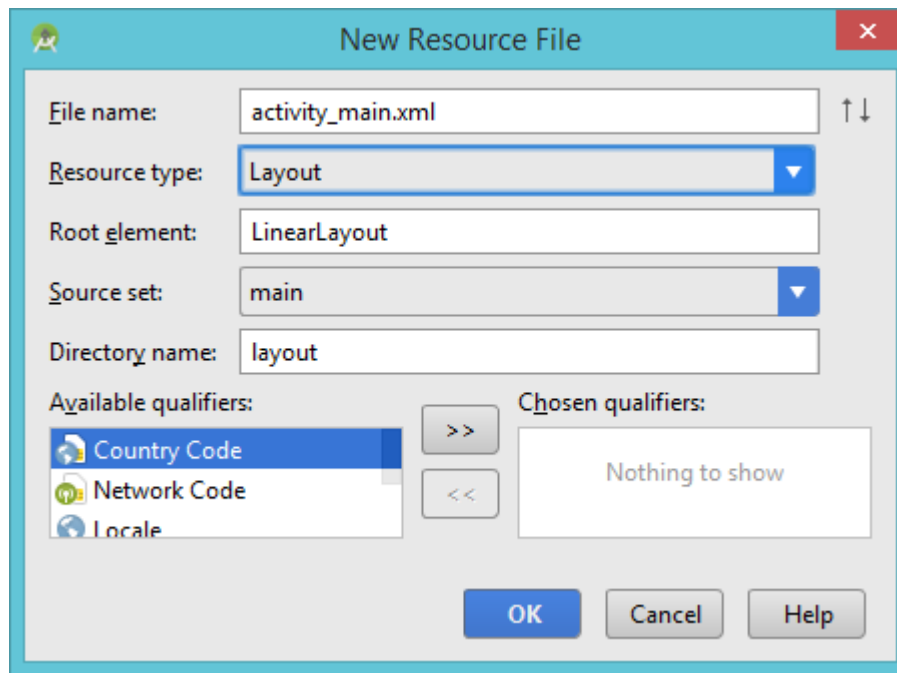


- Renseigner le nom de la classe dans le champ « **Name** ». Clic sur le bouton « **OK** » et la nouvelle classe s'ajoute au projet

## Ajouter un fichier Layout

Un fichier Layout permet de construire l'interface graphique en utilisant le Designer et un code XML. Pour ajouter un fichier Layout, il faut suivre les étapes suivantes :

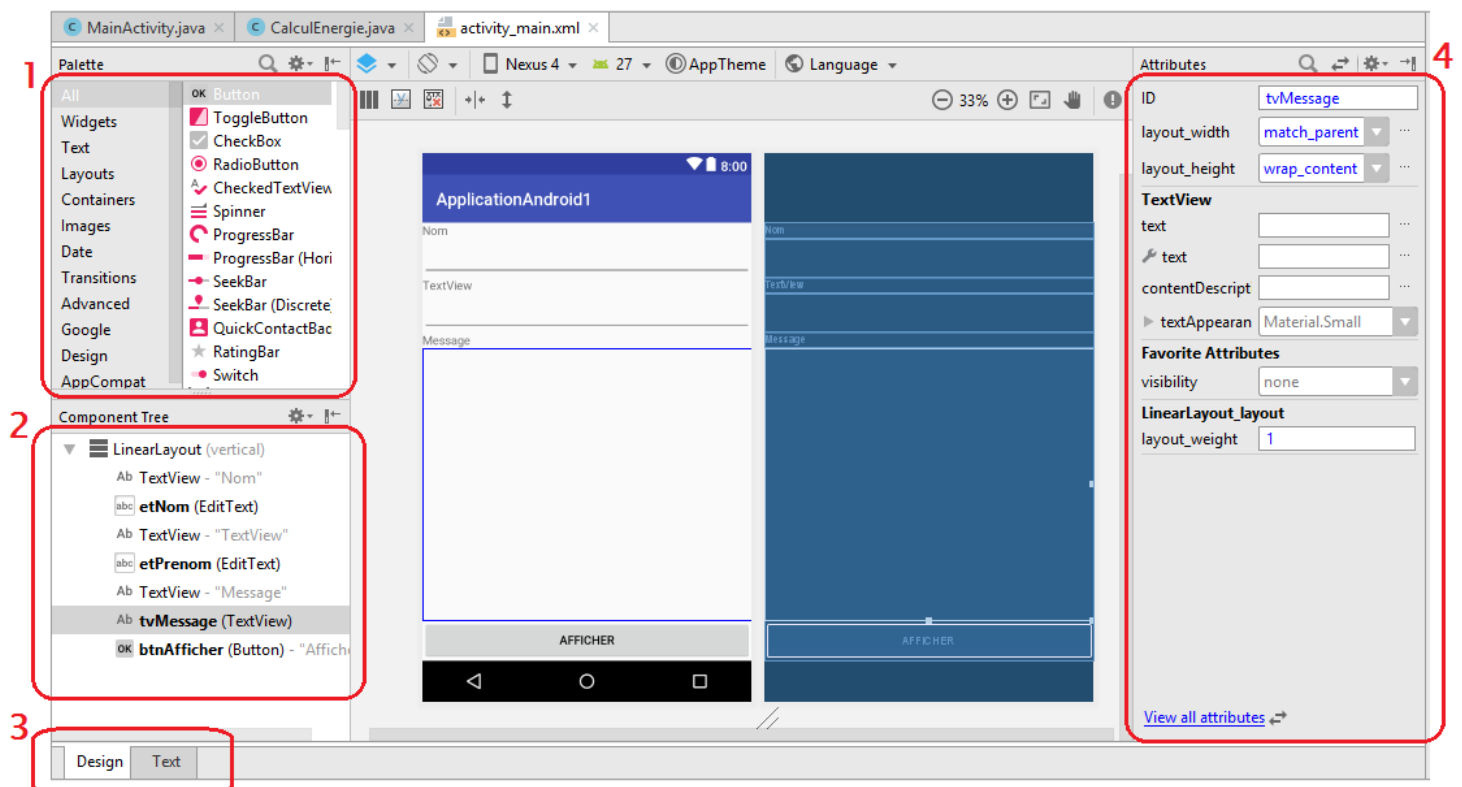
- Clic sur « **File > New > Android Resource File** », et la fenêtre suivante s'ouvre :



- Saisir le nom du fichier Layout dans le champ « **File name** » (**NE PAS** oublier l'extension XML du fichier). A partir du champ « **Resource type** », choisir le type « **Layout** ». Clic sur le bouton « **OK** ». Le nouveau fichier sera ajouter dans le groupe « **app > res > layout** ».

## Construire une interface graphique

Après l'ajout d'un fichier layout, il est possible de créer l'interface graphique avec le designer à partir de la fenêtre suivante :



La **zone 1** contient les différentes vues (composants) qu'on peut ajouter à la GUI.

La **zone 2** contient les différentes vues de la GUI. Il faut sélectionner une vue pour voir et modifier ses attributs à partir de la **zone 4**.

La **zone 3** permet de basculer entre le designer et le code XML du fichier Layout.

La **zone 4** permet de renseigner la valeur des différents attributs (les attributs XML) d'une vue, tel que :

- **ID** (identifiant d'une vue) : permet d'accéder à la vue à partir du code Java. Il est possible de laisser une vue sans identifiant si elle ne sera pas modifiée à partir du code Java.
- **text** : le texte qui sera affiché sur la vue
- **layout\_width** : la largeur de la vue dans son conteneur parent (LinearLayout dans cet exemple)
- **layout\_height** : la hauteur de la vue dans son conteneur parent. Les valeurs possibles pour **layout\_width** et **layout\_height** sont : **wrap\_content** (taille vue = taille de son contenue) et **match\_parent** (taille vue = taille restant du parent : impossible d'ajouter d'autres vues).
- **layout\_weight** (la part d'une vue de son conteneur parent) : si une seule vue a un **layout\_weight** dans un LinearLayout, elle occupe tout l'espace libre du LinearLayout. Sinon, l'espace libre sera partagé selon la part de chaque vue (exemple : **layout\_weight=1** pour vue1 et **layout\_weight=2** pour vue2 implique vue1 occupe 33% et vue2 occupe 66% de l'espace libre).

Par défaut, la **zone 4** affiche les attributs principaux d'une vue. Pour afficher tous les attributs, il faut appuyer sur « **View all attributes** ». Si tous les attributs sont affichés, il est possible d'afficher les attributs principaux en appuyant sur « **View fewer attributes** » (tout en bas de la liste des attributs).

Le designer permet de créer le fichier Layout correspondant à l'interface. Pour consulter le code XML de ce fichier, appuyer sur « Text » à partir de la **zone 3**. Exemple de code :

```
<Button
    android:id="@+id/btnAfficher"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Afficher" />
```

La balise XML « **Button** » contient les différents attributs de la vue « **Button** ».

## d) Les ressources

Les ressources de l'application (disponibles dans le dossier virtuel « **res** ») sont accessibles à partir du code Java à travers la class statique « **R** ». Cette classe est régénérée automatiquement à chaque changement dans les ressources du projet.

**Remarque** : Si l'un des fichiers ressource contient des erreurs, la classe « **R** » devient inaccessible (càd l'appel de la classe « **R** » génère une erreur).

Les ressources sont accessibles à partir du code Java de la manière suivante :

**R.type\_ressource.nom\_ressource**

Il y a plusieurs types de ressources tel que :

- **R.id** : accès par identifiant
- **R.layout** : accès aux fichiers layout
- **R.menu** : accès aux ressources de type menu
- **R.string** : accès aux chaînes de caractères
- Etc. (voir documentation)

Le nom de la ressource peut être :

- le nom du fichier ressource (sans l'extension)
- l'attribut « **android:name** » du fichier XML
- l'identifiant de la ressource (attribut « **android:id** ») si on y accède avec « **R.id** »

La classe « R » retourne l'identifiant de la ressource. Cet identifiant permet ensuite de récupérer l'instance de la ressource en utilisant les méthodes adéquates, telles que :

- La méthode « **getString()** » de la classe « Ressources »
- La méthode « **findViewById()** » de la classe « Activity » : permet d'accéder à des composants graphiques (Vues) à partir de leurs identifiants.

Exemple 1 :

```
Resources res = getResources();  
String title = res.getString(R.string.app_title);
```

Exemple 2 :

```
Button btn = (Button) findViewById(R.id.ok_button);  
Btn.setText("suivant");
```

## 6) Interface utilisateur

### a) Vues et Layouts

Les composants graphiques héritent de la classe « **View** ». On peut regrouper plusieurs composants graphiques dans un « **ViewGroup** ». Un « **ViewGroup** » peut contenir des vues simples et d'autres « **ViewGroup** ». Parmi les « **ViewGroup** » on trouve les layouts qui permettent d'organiser et de positionner les différents composants graphiques.

## Exemples de layouts :

- **LinearLayout** : dispose les éléments de gauche vers la droite ou du haut vers le bas, selon l'orientation du layout (horizontale ou verticale).
- **RelativeLayout** : dispose les éléments les uns par rapport aux autres.
- **TableLayout** : disposition matricielle avec N lignes et M colonnes.
- **GridLayout** : disposition matricielle avec M colonnes et un nombre indéterminé de lignes.
- **ListView** : dispose les éléments dans une liste.

Exemples de vues simples : Button, TextView, EditText, etc.

## b) Les activités (Activity)

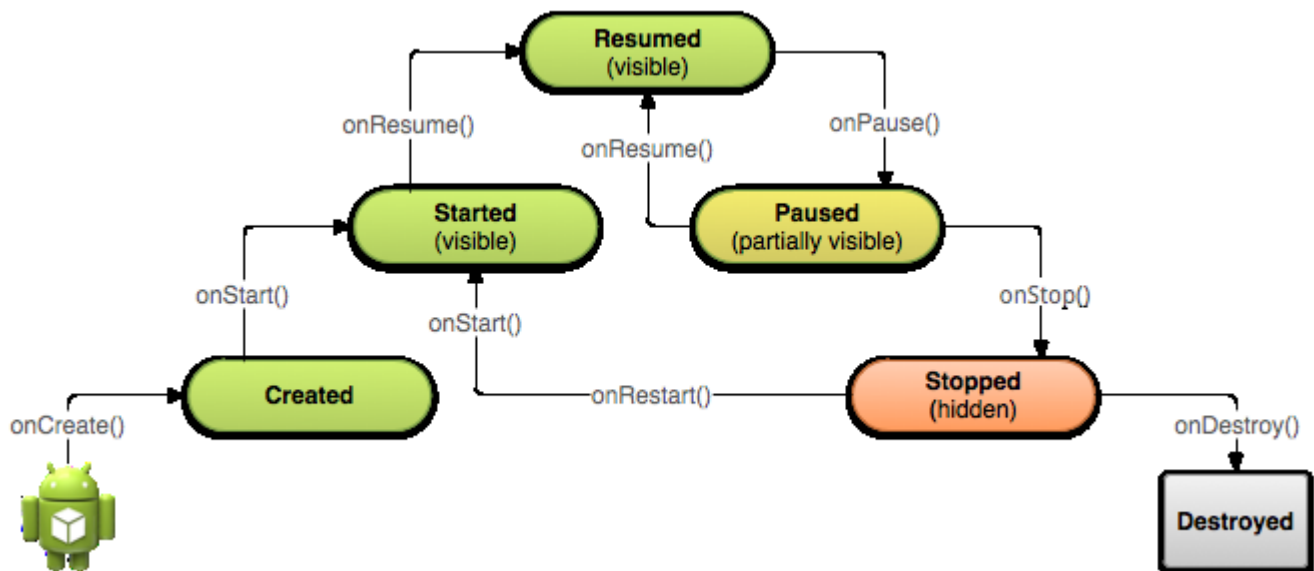
Une application peut contenir une ou plusieurs activités. Le fichier Manifest permet de déclarer l'activité principale qui sera affichée au lancement de l'application (Launcher Activity). Toutes les activités doivent être déclarées dans le manifest.

Par défaut, une activité contient la méthode « onCreate ». Cette méthode est héritée de la classe « Activity ». Elle est appelée lorsque l'activité est créée par le système et entre dans l'état **Created**. Généralement, les opérations effectuées dans cette méthode servent à mettre en place l'interface graphique, à initialiser les variables, à configurer les listeners ou à se connecter à la base de données.

Exemple d'une activité :

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        //your code here  
    }  
}
```

A l'état **Created**, l'activité est créée mais elle n'est pas encore visible. Les différents états d'une activité sont : **Created**, **Started**, **Resumed**, **Paused**, **Stopped** et **Destroyed**.



Différents états d'une activité

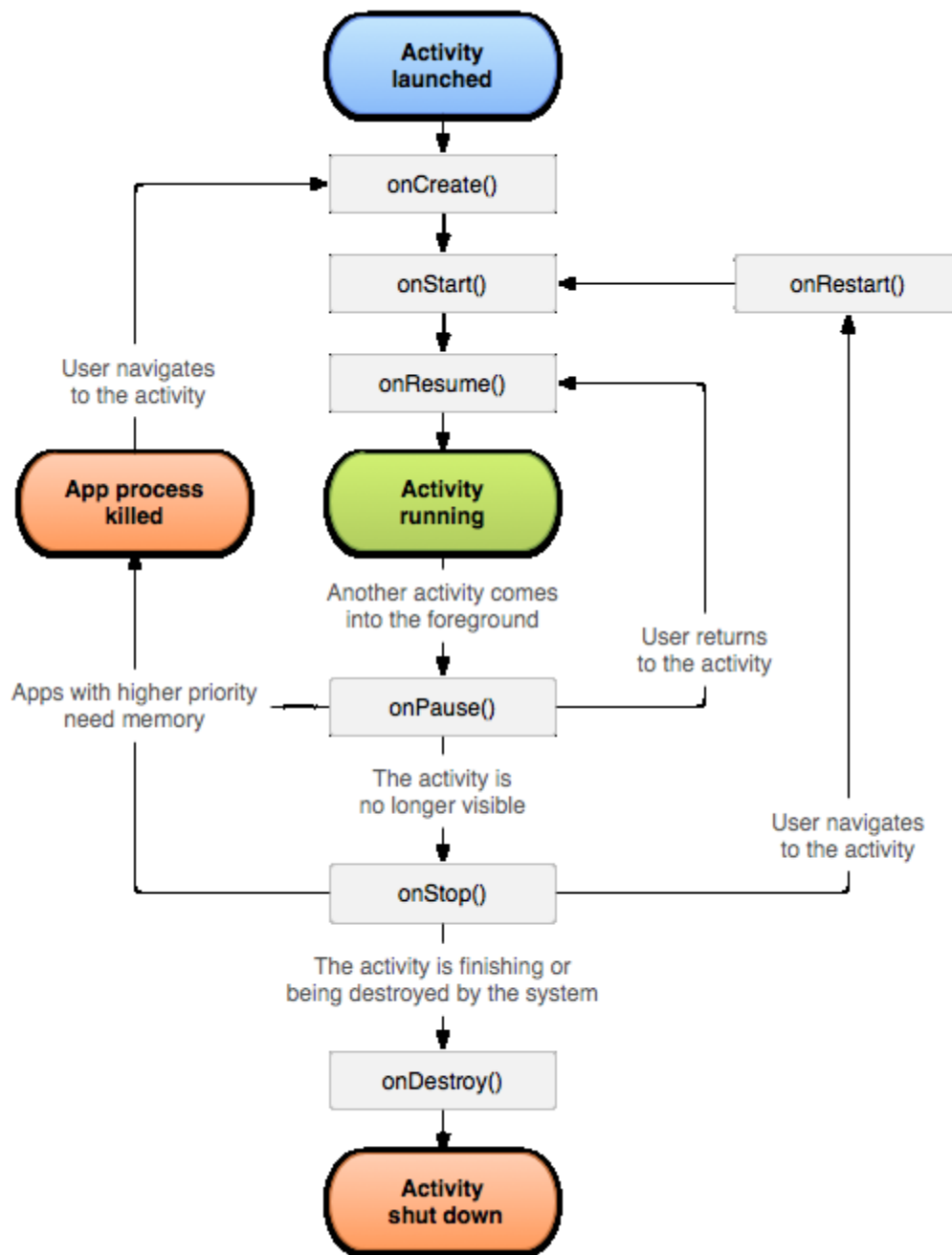
**onStart()** : Cette méthode est appelée par le système lorsque l'activité entre dans l'état **Started**. L'interface graphique devient visible à l'utilisateur, mais il ne peut pas encore interagir avec les différents éléments.

**onResume()** : Cette méthode est appelée lorsque l'activité entre dans l'état **Resumed**. L'activité devient entièrement opérationnelle. L'utilisateur peut utiliser l'application et cliquer sur les différents éléments graphiques. L'application reste dans cet état tant qu'il n'y a pas d'interruption, comme par exemple, la réception d'un appel téléphonique, le démarrage d'une nouvelle activité ou l'affichage d'une boîte de dialogue.

**onPause()** : Cette méthode est appelée lorsque l'activité entre dans l'état **Paused**. Tout ce qui est initié dans `onResume()` doit être mis en pause dans cette méthode. Par exemple, une animation présentée à l'utilisateur est démarrée dans `onResume()` puis stoppée dans `onPause()`.

**onStop()** : Cette méthode est appelée lorsque l'activité entre dans l'état **Stopped**. Par exemple, lorsqu'une nouvelle activité est démarrée, l'activité appelante va se retrouver dans cet état. Elle n'est donc plus visible à l'utilisateur. Les traitements liés à la mise à jour de l'interface graphique peuvent être arrêtés. Les traitements effectués dans cette méthode peuvent être plus importants (comme sauvegarder certaines valeurs dans les `SharedPreferences` par exemple).

**onDestroy()** : Cette méthode est appelée lorsque l'activité est arrêtée. Par exemple, ce peut être après avoir appelée la méthode `finish()`, ou si le système décide d'arrêter l'activité pour libérer de la mémoire.



Activity Lifecycle (<https://developer.android.com/guide/components/activities/activity-lifecycle>)

## Afficher un fichier Layout dans une activité (Activity)

Pour afficher une interface graphique (GUI : Graphical User Interface) dans une activité, on appelle la fonction suivante:

```
setContentView(R.layout.nom_fichier_layout_sans_extension);
```

L'accès à partir d'une activité aux différents composants d'un layout se fait moyennant leurs identifiants. Cela est possible comme suit :

```
findViewById(R.id.idVue); // idVue est l'identifiant renseigné dans
```

```
// l'attribut XML « android:id »
```

### Exemple :

```
EditText etNom = findViewById(R.id.etNom);  
EditText etPrenom = findViewById(R.id.etPrenom);  
TextView tvMessage = findViewById(R.id.tvMessage);
```

## Ajouter un listener à un bouton

Pour ajouter un listener à un bouton, il est possible d'utiliser la fonction

« **setOnClickListener(OnClickListener l)** » d'un bouton comme suit :

```
btnAfficher.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        // code  
    }  
});
```

## Lancement d'une nouvelle activité

Il est possible de lancer une nouvelle activité (nommé activité enfant) à partir de l'activité courante (nommé activité parent). Pour quitter une activité, on peut utiliser la fonction « **finish()** ».

### Sans attendre un résultat

Pour lancer une nouvelle activité sans attendre un résultat, on utilise la classe « **Intent** » et on lui donne le nom de l'activité à lancer. Ensuite on appelle la fonction « **startActivity** ».

Exemple de lancement de l'activité « **AjouterCommandeActivity** » :

```
Intent intent = new Intent(this, AjouterCommandeActivity.class);  
startActivity(intent);
```

Il est possible d'envoyer des données à l'activité qui sera affichée (activité enfant) à travers la méthode « **putExtra** » de l'intent de lancement. Cette méthode reçoit des paires <clé, valeur>. Exemple :

```
Intent intent = new Intent(this, AjouterCommandeActivity.class);  
intent.putExtra("description", "Nouvelle commande pour Tunis");  
intent.putExtra("client", "Ahmed Abid");  
intent.putExtra("numero", 1142512);
```



```
intent.putExtra("paiement", false);  
  
startActivity(intent);
```

Les données envoyées par l'activité parent peuvent être récupérées par l'activité enfant à partir de la méthode « **onCreate** ». Exemple :

```
Intent intent = getIntent();  
String strDesc = intent.getStringExtra("description");  
String strClient = intent.getStringExtra("client");  
int num = intent.getIntExtra("numero", 0);  
boolean payer = intent.getBooleanExtra("paiement", false);
```

La méthode « **intent.getIntExtra** » prend en paramètre, la clé et une valeur par défaut. La valeur par défaut sera retournée si la clé est introuvable. Mais si la clé existe, c'est la valeur correspondante à cette clé qui sera retournée. Même remarque pour les autres méthodes de l'intent qui demandent une clé et une valeur par défaut.

### Avec attente de résultat

Si l'activité parent a besoin d'un résultat de l'activité enfant, il est nécessaire de :

- Côté activité parent :
  - o Lancer l'activité enfant avec la méthode « **startActivityForResult** » (il faut toujours utiliser un intent)
  - o Surcharger la méthode « **onActivityResult** » de l'activité parent qui sera exécutée à la fermeture de l'activité enfant (cette méthode reçoit le résultat de l'activité enfant)
- Côté activité enfant :
  - o L'activité enfant doit sauvegarder les données dans un intent et le retourner au parent avec la méthode « **setResult** ». Avec les données retournées, il faut retourner le résultat (RESULT\_OK ou RESULT\_CANCELED).

Exemple :

#### Côté activité parent :

- Lancement de l'activité enfant :

```
Intent intent = new Intent(this, AjouterCommandeActivity.class);  
startActivityForResult(intent, 0);
```

- Reimplémentation de la méthode « **onActivityResult** » :

```
protected void onActivityResult(int req, int result, Intent data)  
{
```

```

    if (result == RESULT_OK) {
        //code au cas de succès
        //lire données de Intent "data" et faire le traitement
    }
    else {
        //code si annulé
    }
}

```

#### Côté activité enfant :

- Si le résultat est OK (exemple : appui sur le bouton OK de l'activité)

```

void operationOk() {
    Intent intent = new Intent(); // pour sauvegarder les données
    intent.putExtra("operation", 2);
    setResult(RESULT_OK, intent);
    finish();
}

```

- Si l'opération est annulée (exemple : appui sur le bouton Annuler de l'activité)

```

void operationAnnuler() {
    Intent intent = new Intent();
    setResult(RESULT_CANCELED, intent);
    finish();
}

```

### c) Toast

Permet d'afficher un message pour une courte ou longue durée.

Exemple :

```

Toast.makeText(this, "Bonjour", Toast.LENGTH_LONG).show();
Toast.makeText(this, "Bonjour", Toast.LENGTH_SHORT).show();

```

### d) ListView

Une ListView est un groupe de vues qui affiche une liste déroulante d'items. Les items sont insérés dans la liste moyennant une instance d'une classe qui hérite de **android.widget.Adapter** (exemple : **ArrayAdapter** et **SimpleAdapter**). **ArrayAdapter** permet de créer une liste simple de labels (les items sont des labels).

#### Exemple 1 : création d'une ListView

```

ArrayList listEtudiants = new ArrayList();
listEtudiants.add("Ali Ben Brahim");
listEtudiants.add("Anis Dridi");
listEtudiants.add("Imen Heni");

```

```

ArrayAdapter adapter = new ArrayAdapter(this,
    android.R.layout.simple_list_item_1, listEtudiants);
ListView lvEtudiants = findViewById(R.id.lvEtudiants);
lvEtudiants.setAdapter(adapter);

```

### **Exemple 2 : mise à jour de la ListView**

```

listEtudiants.add("Imen Heni");
adapter.notifyDataSetChanged();

```

**SimpleAdapter** permet de créer une ListView avec des items plus compliqués. Il est possible de créer des items personnalisés en créant un adapter personnalisé (càd une classe qui hérite de **android.widget.Adapter**).

### **Exemple 3 : création d'un Adapter personnalisé**

```

public class CustomArrayAdapter extends ArrayAdapter {
    ArrayList listValues;
    LayoutInflater inflater;

    public CustomArrayAdapter(Context context,
        ArrayList listValues,
        LayoutInflater inflater) {
        super(context, -1, listValues);
        this.listValues = listValues;
        this.inflater = inflater;
    }

    public View getView(int position, View convertView,
        ViewGroup parent) {

        View view = inflater.inflate(R.layout.list_row,
            parent, false);

        ImageView imgView = (ImageView)
            view.findViewById(R.id.listRowImageView);
        TextView item1 = (TextView)
            view.findViewById(R.id.listRowTextFirst);
        TextView item2 = (TextView)
            view.findViewById(R.id.listRowTextSecond);

        Vector itemValues = (Vector) listValues.get(position);
        String nom, prenom, formation;
        nom = (String) itemValues.get(0);
        prenom = (String) itemValues.get(1);
        formation = (String) itemValues.get(2);

        item1.setText(nom + " " + prenom);
        item2.setText(formation);
        imgView.setImageResource(R.mipmap.ic_launcher);

        return view;
    }
}

```

```

    }
}

//instantiation
ArrayList listEtudiants = new ArrayList();
Vector itemValues = new Vector();
itemValues.add(nom);
itemValues.add(prenom);
itemValues.add(formation);
listEtudiants.add(itemValues);

CustomArrayAdapter cAdapter = new CustomArrayAdapter(this,
    listEtudiants, getLayoutInflater());

ListView lvEtudiants = findViewById(R.id.lvEtudiants);
lvEtudiants.setAdapter(cAdapter);

```

#### **Exemple 4 : gestion des événements**

//appui simple :

```

lvEtudiants.setOnClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view,
        int position, long id) {
        Vector itemValues = (Vector)
            lvEtudiants.getItemAtPosition(position);

        System.out.println("val1=" + itemValues.get(0) +
            ", val2=" + itemValues.get(1) +
            ", val3=" + itemValues.get(2));
    }
});

```

//appui long :

```

lvEtudiants.setOnItemLongClickListener(
    new AdapterView.OnItemLongClickListener() {
    @Override
    public boolean onItemLongClick(AdapterView<?> parent, View view,
        int position, long id) {
        System.out.println("Long click : pos=" + position);

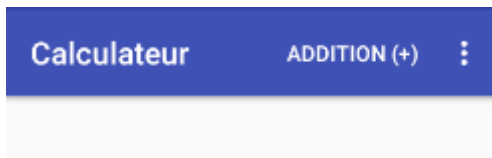
        return true;
    }
});

```

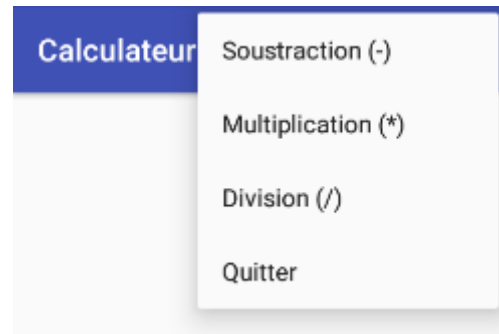
#### **d) Menu**

Voir <https://developer.android.com/guide/topics/ui/menus>

Un menu s'affiche dans la barre de titre (ActionBar) de l'activité. Un menu contient des « items ». Un item peut s'afficher sur la barre de titre ou peut être masqué. Pour afficher les items masqués, on appuie sur les « 3 points verticaux » du menu. Exemple de menu :



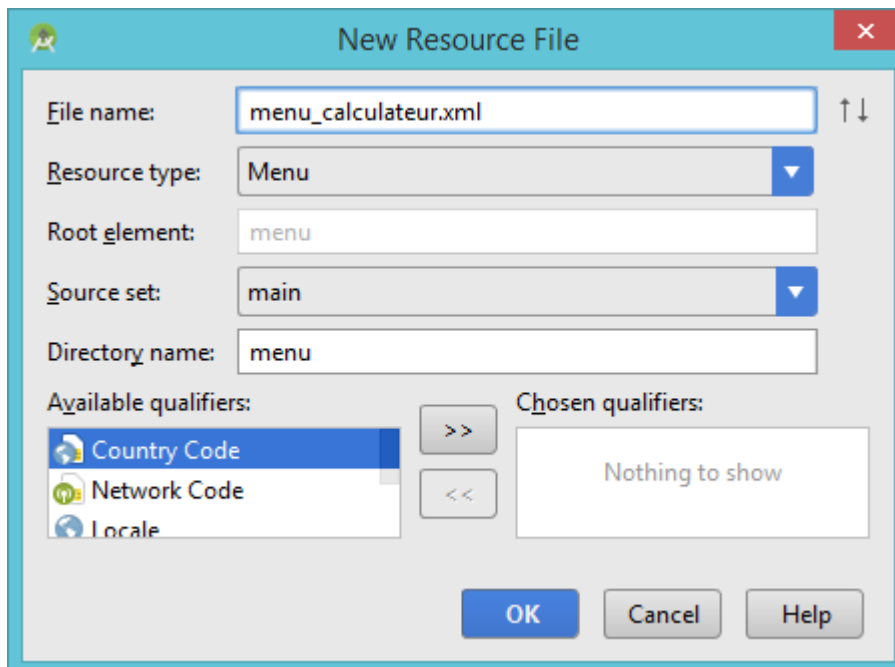
L'item « ADDITION (+) » est affiché sur la barre de titre, et les autres items du menu sont accessibles à travers les « 3 points verticaux »



Les items masqués du menu

Il est possible de construire le menu d'une activité en utilisant un fichier ressource de type « **Menu** » et le Designer. Pour ajouter un fichier ressource de type « **Menu** », il faut suivre les étapes suivantes :

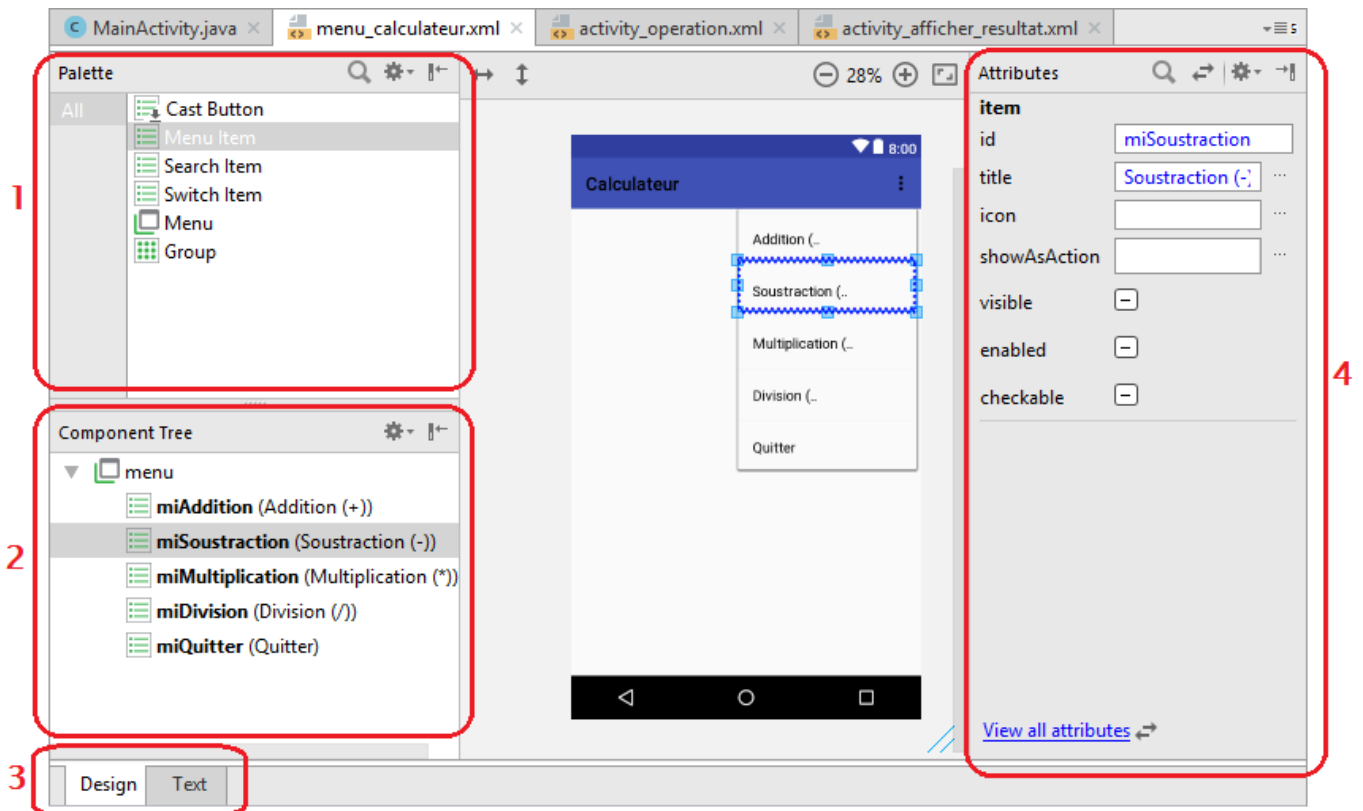
- Clic sur « **File > New > Android Resource File** », et la fenêtre suivante s'ouvre :



- Saisir le nom du fichier Menu dans le champ « **File name** » (**NE PAS** oublier l'extension XML du fichier). A partir du champ « **Resource type** », choisir le type « **Menu** ». Clic sur le bouton « **OK** ». Le nouveau fichier sera ajouté dans le groupe « **app > res > menu** ».

## Construire un Menu

Après l'ajout d'un fichier Menu, il est possible de créer un menu avec le designer à partir de la fenêtre suivante :



La **zone 1** contient les différents composants disponibles. Un « **Menu Item** » est un item qu'on peut ajouter à un menu.

La **zone 4** permet de renseigner la valeur des différents attributs d'un item, tel que :

- **ID** (identifiant) : permet d'accéder à un item à partir du code Java
- **title** : le texte de l'item
- **showAsAction** : indique comment l'item sera affiché dans la barre de titre. Exemple de valeurs possibles sont : **never** (l'item sera toujours masqué), **ifRoom** (afficher l'item sur la barre de titre s'il y a de l'espace), **always** (l'item sera toujours affiché dans le barre de titre).

### Exemple 1 : menu simple

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:title="Nouveau"
          android:id="@+id/miNouveau"
          android:icon="@drawable/icNouveau"
          app:showAsAction="never" />
    <item android:title="Supprimer"
          android:id="@+id/miSupprimer" />
</menu>
```

### Exemple 2 : menu avec un sous-menu

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:title="Nouveau"
        android:id="@+id/miNouveau" />

    <item android:title="Supprimer"
        android:id="@+id/miSupprimer">
        <!-- "Nouveau" submenu -->
        <menu>
            <item android:title="Supprimer Tous"
                android:id="@+id/miSupprimerTous" />
            <item android:title="Supprimer Selection"
                android:id="@+id/miSupprimerSelection" />
        </menu>
    </item>
</menu>
```

### Afficher le menu dans l'activité

Pour afficher le menu du fichier ressource dans l'activité, il faut surcharger la méthode héritée « **onCreateOptionsMenu** ». Cette méthode est appelée par le système au moment de la création du menu de l'activité.

#### Exemple :

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater mi = getMenuInflater();
    mi.inflate(R.menu.menu_calculateur, menu);

    return true;
}
```

### Ajouter un listener aux items :

Lors de l'appui sur un item, le system appelle la méthode « **onOptionsItemSelected** » et lui fournit l'item sélectionné. Il faut redéfinir cette méthode pour gérer les évènements lors de l'appui sur les items.

Exemple :

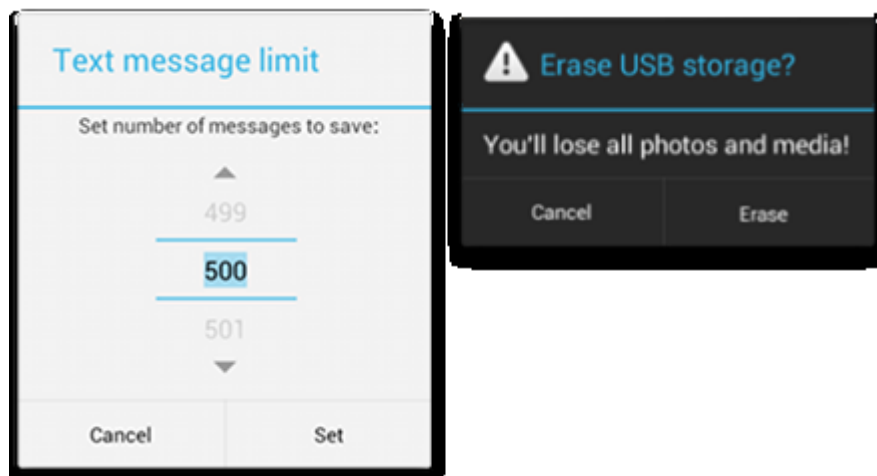
```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch(item.getItemId()) {
        case R.id.miAddition:
            Toast.makeText(this, "item +", Toast.LENGTH_LONG).show();
            this.operation = 1;
            break;
        case R.id.miSoustraction:
            Toast.makeText(this, "item -", Toast.LENGTH_LONG).show();
            this.operation = 2;
            break;
        case R.id.miQuitter:
            finish();
            break;
    }

    return true;
}
```

## e) Dialog

Voir <https://developer.android.com/guide/topics/ui/dialogs>

Un « **Dialog** » est une petite fenêtre qui demande à l'utilisateur de faire un choix ou de saisir des informations complémentaires. Un « **Dialog** » ne couvre pas la totalité de l'écran.



Exemple de classes dérivées de « **Dialog** » ayant une UI prédéfinie :

- **AlertDialog** : A dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout.
- **DatePickerDialog** or **TimePickerDialog** : A dialog with a pre-defined UI that allows the user to select a date or time.



### **Exemple : Dialog avec une liste de choix composée par « Supprimer » et « Détails »**

```
void buildAndShowDialog(int position) {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);

    final Vector itemValues = (Vector)
        lvEtudiants.getItemAtPosition(position);
    builder.setTitle((String) itemValues.get(0));

    builder.setItems(new String[]{"Supprimer", "Détailles"},
        new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                if (which == 0) {
                    //bouton Supprimer
                    System.out.println("Supprimer");
                }
                else if (which == 1) {
                    //bouton Détails
                    System.out.println("Détailles");
                }
            }
        });

    builder.show();
}
```

### **f) Fragment**

**Voir :** <https://developer.android.com/guide/components/fragments>

<https://developer.android.com/training/basics/fragments/communicating>

Un fragment est une portion de l'interface graphique d'une activité.

Un fragment doit obligatoirement se trouver dans une activité.

Une activité peut contenir 0 ou plusieurs fragments.

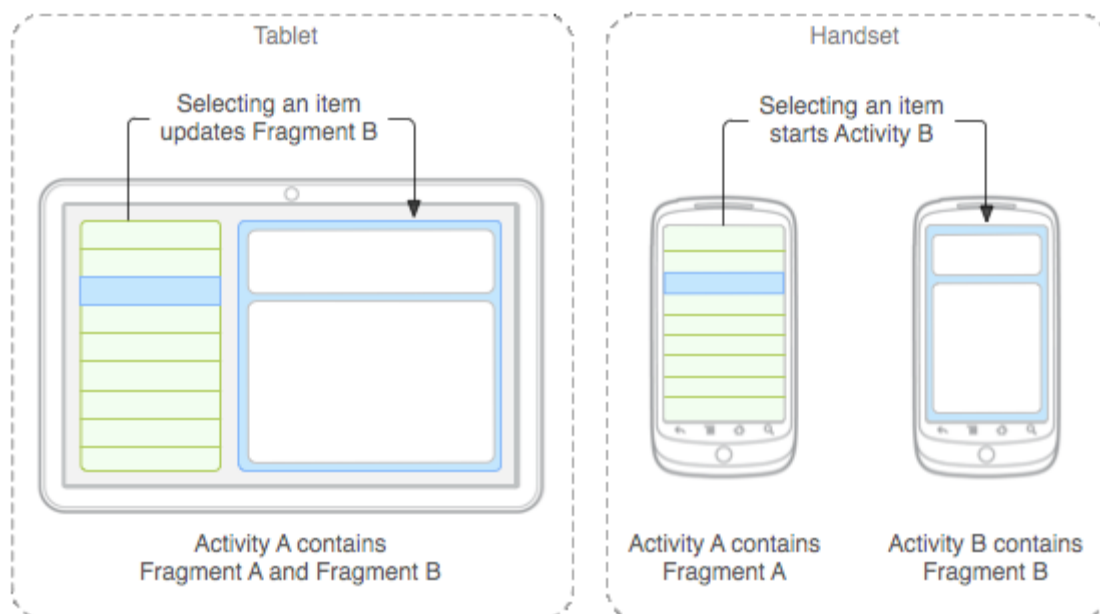
Un fragment a son propre cycle de vie qui dépend du cycle de vie de l'activité hôte : lorsque l'activité est à état Paused, tous ses fragments entre dans cet état, et lorsque l'activité est détruite, tous ses fragments le sont également.

Lorsque l'activité hôte est en cours d'exécution (état Resumed), il est possible de manipuler ses fragments indépendamment les uns des autres. Exemple : ajout et suppression des fragments

L'objectif des fragments est l'adaptation d'une application Android à toutes les tailles d'appareils existants.

Par exemple, une application peut incorporer deux fragments (liste des articles et contenu de l'article sélectionné) dans l'activité A, lorsqu'elle est exécutée sur une tablette. Toutefois, sur un smartphone, l'espace disponible est insuffisant pour afficher les deux fragments. Ainsi l'activité A n'inclut que le

fragment de la liste d'articles. Lorsque l'utilisateur sélectionne un article, on démarre l'activité B, qui contient le second fragment permettant de lire l'article sélectionné. Ainsi, l'application supporte les tablettes et les smartphones en réutilisant des fragments dans différentes combinaisons (voir figure).



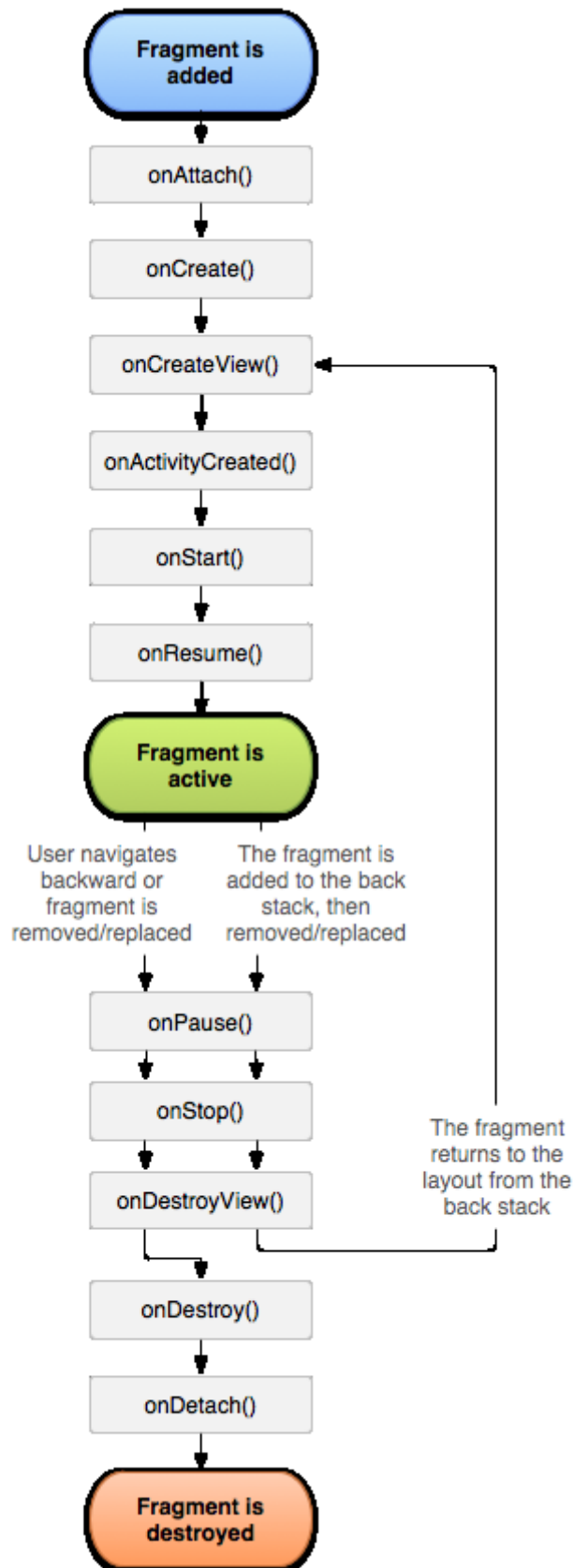
**Remarque :** sans les fragments il était déjà possible de s'adapter à toutes les tailles d'écrans grâce aux layouts. Par contre, l'activité qui contrôlait tous ces cas devenait très complexe. Solution : les fragments.

### Cycle de vie d'un fragment :

La classe `Fragment` a des méthodes de rappel (callback methods) qui ressemblent aux méthodes d'une activité, telles que : **`onCreate()`**, **`onStart()`**, **`onPause()`**, et **`onStop()`**. En général, lorsqu'on crée un nouveau fragment, on implémente souvent les méthodes suivantes :

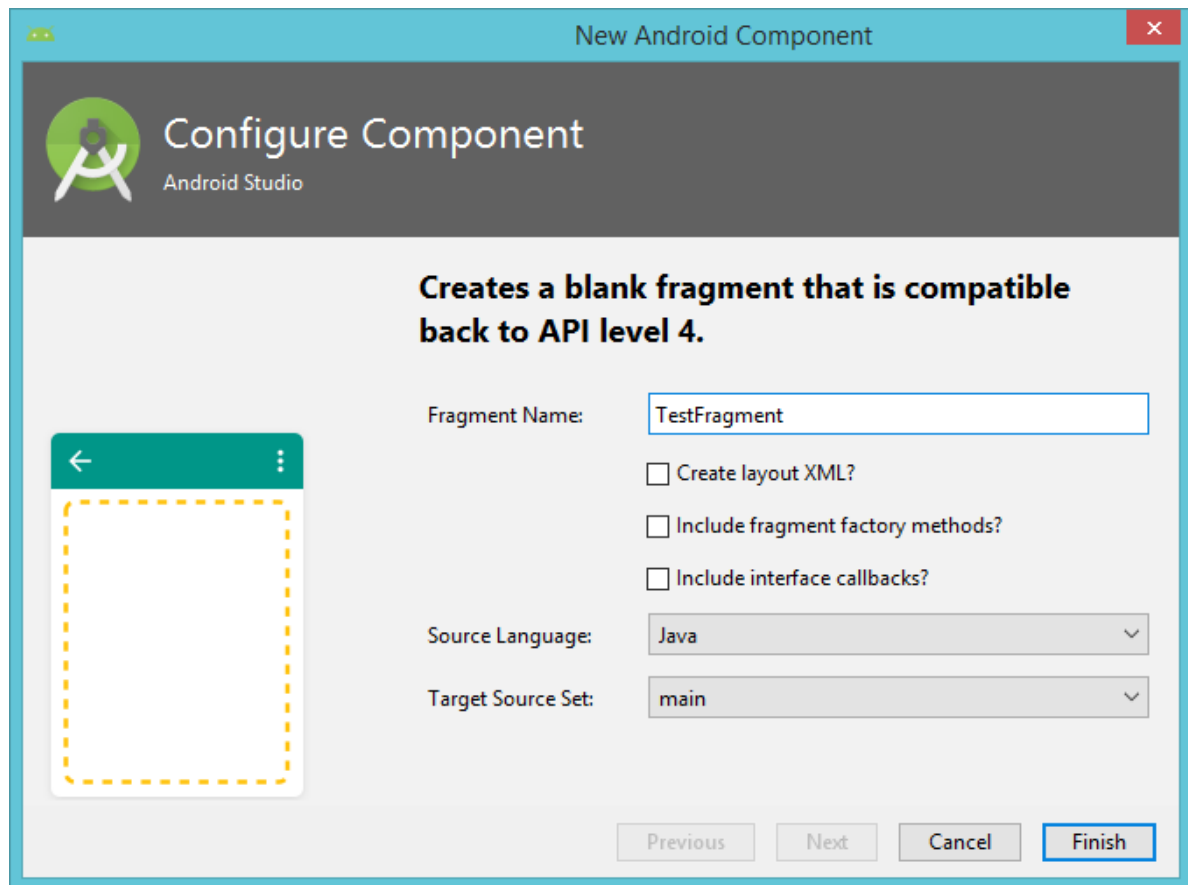
- **`onCreate()`** : le système appelle cette méthode au moment de création du fragment
- **`onCreateView()`** : le système appelle cette méthode lorsque le fragment doit afficher son interface utilisateur pour la première fois
- **`onDestroy()`** : le système appelle cette méthode lorsque le fragment est détruit.

Plusieurs autres méthodes existent et peuvent être utiles dans les différentes phases du cycle de vie d'un fragment (voir figure suivante).



Pour ajouter un nouveau fragment au projet :

- Sélectionner le module « app »
- File > New > Fragment > Fragment (Blank)
- Renseigner le nom du fragment et les autres paramètres de la fenêtre suivante :



Le code source obtenu est :

```
package fsm.tpfragment;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class TestFragment extends Fragment {

    public TestFragment() {
        // Required empty public constructor
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        TextView textView = new TextView(getActivity());
        textView.setText(R.string.hello_blank_fragment);
        return textView;
    }
}
```

Il faut utiliser la méthode « **onCreateView** » pour définir un layout.

Exemple :

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_test,
                                container, false);

    Button btn;
    btn = view.findViewById(R.id.fragbutton);

    btn.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            System.out.println("bonjour");
        }
    });

    return view;
}
```

Ajouter le fragment dans l'activité :

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    TestFragment fragment = new TestFragment();

    FragmentManager fragmentManager = getSupportFragmentManager();
    FragmentTransaction ft = fragmentManager.beginTransaction();
    ft.add(R.id.llFragmentContainer, testFragment);
    ft.commit();
}
```

Remplacer un fragment par un autre

```
FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
ft.replace(R.id.llFragmentContainer, confirmFragment);
ft.commit();
```

Masquer/Afficher un fragment

```
FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
ft.hide(testFragment);
ft.commit();

//...
FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
ft.show(testFragment);
ft.commit();
```

## Supprimer un fragment

```
FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
ft.remove(testFragment);
ft.commit();
```

## Communication entre Activité et Fragment

Une activité peut envoyer des données à un fragment avec 2 méthodes principales :

- En utilisant un objet « **Bundle** » : on ajoute les données dans une instance de la classe « **Bundle** », ensuite on fournit cette instance au fragment moyennant la méthode « **setArguments** ». Finalement, le fragment est capable de lire les données de l'objet « **Bundle** ».
- En appelant les méthodes publiques du fragment.

### Exemple :

```
//activité
@Override
protected void onCreate(Bundle savedInstanceState) {

    //...

    testFragment = new TestFragment();
    Bundle params = new Bundle();
    params.putString("nom", "Dridi");
    params.putString("prenom", "Ali");
    testFragment.setArguments(params);

    //...

}

//fragment
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {

    //...

    Bundle params = getArguments();
    String nom = params.getString("nom");
    String prenom = params.getString("prenom");

    //...

}
```

Un fragment peut communiquer avec son activité en utilisant une interface. Cette interface doit contenir les méthodes que le fragment doit exécuter. Elle doit être implémentée par la suite dans l'activité (càd les méthodes de l'interface doivent être implémentées dans l'activité). Le fragment récupère l'implémentation de l'interface (càd une instance de l'activité qui implémente l'interface) et peut ensuite appeler les méthodes de l'interface qui sont implémentées dans l'activité. En résumé, un fragment peut communiquer avec son activité en réalisant les étapes suivantes :

- Créer une interface : cette interface contient les méthodes que le fragment doit exécuter
- Implémenter l'interface dans l'activité
- Le fragment récupère l'implémentation de l'interface et exécute les méthodes de l'interface selon le besoin

#### Exemple :

```
//1) Créer une interface
public interface OnFragmentEventListener {
    public void ajouterEtudiant(String nom, String prenom);
    public void annulerOperation();
}

//2) implémenter l'interface dans l'activité
public class MainActivity extends AppCompatActivity implements
OnFragmentEventListener {
    //...
    //...
    @Override
    public void ajouterEtudiant(String nom, String prenom) {
        System.out.println("Ajouter " + nom + " " + prenom);

        FragmentTransaction ft =
            getSupportFragmentManager().beginTransaction();
        ft.replace(R.id.llFragmentContainer, confirmFragment);
        ft.commit();
    }

    @Override
    public void annulerOperation() {
        System.out.println("Annuler");

        FragmentTransaction ft =
            getSupportFragmentManager().beginTransaction();
        ft.replace(R.id.llFragmentContainer, confirmFragment);
        ft.commit();
    }
}
```

```

//3) le fragment
public class TestFragment extends Fragment {
    OnFragmentEventListener listener;
    EditText etNom, etPrenom;

    public TestFragment() {
        // Required empty public constructor
    }

    @Override
    public View onCreateView(LayoutInflater inflater,
                             ViewGroup container, Bundle savedInstanceState) {

        View view = inflater.inflate(R.layout.fragment_test,
                                     container, false);

        etNom = view.findViewById(R.id.ft_et_nom);
        etPrenom = view.findViewById(R.id.ft_et_prenom);

        listener = (OnFragmentEventListener) getActivity();

        Button btnAjouter = view.findViewById(R.id.ft_btn_ajouter);
        btnAjouter.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                System.out.println("Ajouter");
                listener.ajouterEtudiant(etNom.getText().toString(),
                                       etPrenom.getText().toString());
            }
        });

        Button btnAnnuler = view.findViewById(R.id.ft_btn_annuler);
        btnAnnuler.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                System.out.println("Annuler");
                listener.annulerOperation();
            }
        });

        return view;
    }
}

```

## 7) Persistance des données

Android fournit plusieurs méthodes de stockage des données, telles que :

- Données actives d'une activité (Bundle)
- Fichier ressources read-only (répertoire res)
- Préférences partagées (SharedPreferences)



- Système de fichiers
  - o Donnée interne de l'application (mémoire flash) ou données en cache (résultat calcul intermédiaire)
  - o Donnée externe partagée (carte SD)
  - o En ligne sur le cloud
- Base de données (SQLite)

## a) Préférences partagées

Voir : <https://developer.android.com/training/data-storage/shared-preferences>

Les préférences partagées (SharedPreferences) permettent de sauvegarder des données sous forme de paires « clé, valeur ». Un objet SharedPreferences utilise un fichier contenant des paires « clé, valeur », et fourni des méthodes simples permettant de lire et d'écrire ces paires.

Il est possible d'obtenir un objet SharedPreferences avec les méthodes suivantes :

- **getSharedPreferences()** : cette méthode permet d'accéder à un fichier de préférences partagées identifié par son nom. Le nom du fichier doit être unique. La méthode peut être appelée à partir de n'importe quel Context de l'application (Activity, Fragment, etc.) et prend 2 paramètres : nom du fichier et le mode d'accès.

```
SharedPreferences preferences =
    getSharedPreferences("fsm.tp5.prefs1", Context.MODE_PRIVATE);
```

- **getPreferences()** : cette méthode doit être utilisée à partir d'une activité et permet d'accéder à un seul fichier de préférences partagées. La méthode ne demande pas le nom du fichier car c'est le fichier par défaut appartenant à l'activité.

```
SharedPreferences preferences =
    getPreferences(Context.MODE_PRIVATE);
```

- **getDefaultSharedPreferences()** : cette méthode permet d'accéder à un fichier par défaut propre à une application. Elle permet à tous les composants d'une application (activités, fragments, etc.) de partager un fichier par défaut de préférences partagées. C'est une méthode statique de la classe « PreferenceManager ». Elle prend un seul paramètre qui est le contexte.

```
SharedPreferences preferences =
    PreferenceManager.getDefaultSharedPreferences(this);
```

## Ecriture dans les préférences partagées :

Pour écrire, il faut créer un objet « SharedPreferences.Editor » et utiliser les méthodes de cet objet, telles que : **putInt()** et **putString()**. Ces méthodes prennent les paires « clé, valeur ». Finalement, il faut appeler la méthode « **commit()** » ou « **apply()** » pour sauvegarder les modifications.

### Exemple :

```
SharedPreferences.Editor editor = preferences.edit();
editor.putString("username", etUser.getText().toString());
editor.putString("title", etTitle.getText().toString());
editor.putString("color", etColor.getText().toString());
editor.putBoolean("show_username", true);
editor.commit();
```

### Lecture des préférences partagées :

Pour lire à partir d'un fichier de préférences partagées, il faut utiliser les méthodes de lecture (**getInt()**, **getString()**, etc.) en fournissant la clé, et si nécessaire, une valeur par défaut à retourner si la clé est indéfinie.

### Exemple :

```
String username = preferences.getString("username", "no username");
String title = preferences.getString("title", "no title");
String bgColor = preferences.getString("color", "no color");
boolean showUsername = preferences.getBoolean("show_username", false);
```

**Remarque :** Afin de permettre à l'utilisateur de consulter et renseigner ses préférences, il faut lui créer une interface graphique de saisie (fichier layout, fragment et/ou activité, listener, etc.). Une autre alternative est d'utiliser la librairie « **Preference** ».

**Voir :** <https://developer.android.com/guide/topics/ui/settings/>

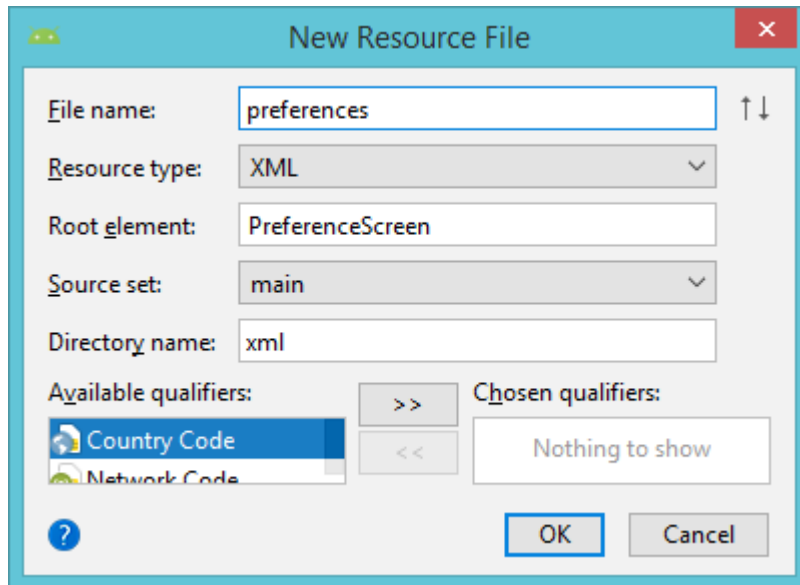
### Création d'une interface de saisie avec la librairie « Preference »

Pour créer une interface de saisie avec la librairie « **Preference** », il faut suivre les étapes suivantes :

1. Créer un fichier ressource de type XML (dossier res/xml) ayant « **PreferenceScreen** » comme racine (root).
2. Compléter ce fichier avec le designer en y ajoutant les composants nécessaires
3. Créer un fragment qui hérite de la classe « PreferenceFragment » et implémenter la méthode « onCreate() » afin d'afficher le fichier XML dans le fragment (en appelant la méthode « **addPreferencesFromResource()** »).
4. Créer une nouvelle activité permettant d'afficher le fragment de préférences. Cette activité peut ensuite être appelée à partir de l'activité principale pour consulter et modifier les préférences.

**Remarque :** La librairie « Preference » permet de consulter et de modifier les préférences partagées d'une application à partir d'une interface graphique. Ces préférences sont sauvegardées dans le fichier par défaut de l'application. Afin de consulter ces préférences avec un code Java, il faut créer l'objet **SharedPreferences** avec la méthode « **getDefaultSharedPreferences()** ».

**Etape 1 :** Création d'un fichier ressource de type XML, nommé « **preferences.xml** » ayant comme racine (root) la valeur « **PreferenceScreen** » :



**Etape 2 :** Compléter le fichier ressource avec le designer en y ajoutant les composants nécessaires, tels que EditTextPreference et CheckBoxPreference. **Exemple :**

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">

    <EditTextPreference
        android:defaultValue="anonyme"
        android:dependency="show_username"
        android:key="username"
        android:selectAllOnFocus="true"
        android:singleLine="true"
        android:summary="Nom de l'utilisateur"
        android:title="Nom" />
    <CheckBoxPreference
        android:defaultValue="false"
        android:key="show_username"
        android:summary="Afficher nom utilisateur"
        android:title="Afficher" />
    <EditTextPreference
        android:defaultValue="Gestionnaire"
        android:key="title"
        android:selectAllOnFocus="true"
        android:singleLine="true"
        android:summary="Titre de l'application"
        android:title="Titre" />
    <EditTextPreference
        android:defaultValue="blanc"
        android:key="color"
        android:selectAllOnFocus="true"
        android:singleLine="true"
        android:summary="Couleur arrière plan"
        android:title="Couleur" />
</PreferenceScreen>
```

**Etape 3 :** Création du fragment « MyPreferencesFragment » héritant de la classe « PreferenceFragment » et permettant l’affichage du fichier ressource « preferences.xml » :

```
public class MyPreferencesFragment extends PreferenceFragment {  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        addPreferencesFromResource (R.xml.preferences);  
    }  
}
```

**Etape 4 :** Création de l’activité permettant l’affichage du fragment « MyPreferencesFragment »

```
public class MyPreferencesActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setTitle("Preferences utilisateur");  
  
        MyPreferencesFragment fragment = new MyPreferencesFragment();  
        FragmentTransaction ft =  
            getSupportFragmentManager().beginTransaction();  
        ft.add(android.R.id.content, fragment).commit();  
    }  
}
```

## Résultat d’exécution

Preferences utilisateur	Preferences utilisateur
Nom Nom de l'utilisateur	Nom Nom de l'utilisateur
Afficher Afficher nom utilisateur <input checked="" type="checkbox"/>	Afficher Afficher nom utilisateur <input type="checkbox"/>
Titre Titre de l'application	Titre Titre de l'application
Couleur Couleur arrière plan	Couleur Couleur arrière plan



## b) Base de données SQLite

Création de l'objet « SQLiteDatabase »

```
SQLiteDatabase sqlDb = openOrCreateDatabase("mydatabase.db",
                                           SQLiteDatabase.OPEN_READWRITE, null);
```

Exécution d'une requête qui ne retourne aucun résultat (création d'une table si inexistante)

```
sqlDb.execSQL("CREATE TABLE IF NOT EXISTS contact (id INTEGER "
              + " PRIMARY KEY AUTOINCREMENT, nom VARCHAR, prenom VARCHAR"
              + ", telephone VARCHAR, email VARCHAR);");
```

Insertion d'une ligne dans une table

```
ContentValues values = new ContentValues();
values.put("nom", nom);
values.put("prenom", prenom);
values.put("telephone", telephone);
values.put("email", email);

long id = sqlDb.insert("contact", null, values);
//sqlDb.insert() retourne -1 au cas d'erreur
if (id == -1) {
    Toast.makeText(this, "Erreur d'insertion dans la base",
                  Toast.LENGTH_SHORT).show();
}
```

## Modification d'une ligne dans une table

```
ContentValues newValues = new ContentValues();
newValues.put("telephone", "77123456");
newValues.put("email", "ali@gmail.com");

sqlDb.update("contact", newValues, "id=1", null);
```

```
ContentValues newValues = new ContentValues();
newValues.put("telephone", "77123456");
newValues.put("email", "ali@gmail.com");

String args[] = new String[]{"Dridi", "Ali"};
sqlDb.update("contact", newValues, "nom=? AND prenom=?", args);
```

```
sqlDb.execSQL("UPDATE contact SET telephone='77123456',
email='ali@gmail.com' WHERE nom='Dridi' AND prenom='Ali'");
```

## Lecture des résultats d'une requête

```
Cursor resultSet = sqlDb.rawQuery("SELECT * FROM contact", null);
if (resultSet.moveToFirst()) {
    //lire une ligne et verifier s'il y en a d'autres
    do {
        String id = resultSet.getString(0);
        String nom = resultSet.getString(1);
        String prenom = resultSet.getString(2);
        String telephone = resultSet.getString(3);
        String email = resultSet.getString(4);
        String text = id + " - " + nom + " " + prenom
                    + " - " + telephone + " " + email;

        System.out.println(text);
    } while (resultSet.moveToNext());
}

resultSet.close();
```

## Suppression d'une ou plusieurs lignes

```
sqlDb.delete("contact", "id=1", null);
```

```
String args[] = new String[]{"Dridi", "Ali"};
sqlDb.delete("contact", "nom=? AND prenom=?", args);
```

```
//supprimer toutes les lignes
sqlDb.delete("contact", "1", null);
```

## 8) Mise en place de la navigation

### a) Création de vues de balayage avec des onglets

Voir : <https://developer.android.com/training/implementing-navigation/lateral>

Les vues par balayage permettent une navigation latérale entre les écrans frères, tels que les onglets avec un geste du doigt horizontal (un motif parfois appelé pagination horizontale).

### b) Création d'un Navigation Drawer ( tiroir de navigation)

Voir : <https://developer.android.com/training/implementing-navigation/nav-drawer>

Le tiroir de navigation est un panneau d'interface utilisateur qui affiche le menu de navigation principal de l'application. Il est masqué lorsqu'il n'est pas utilisé, mais apparaît lorsque l'utilisateur fait glisser un doigt du côté gauche de l'écran ou, lorsqu'il se trouve au niveau supérieur de l'application, l'utilisateur

touche l'icône du tiroir  dans la barre d'action.

#### Ajouter des dépendances

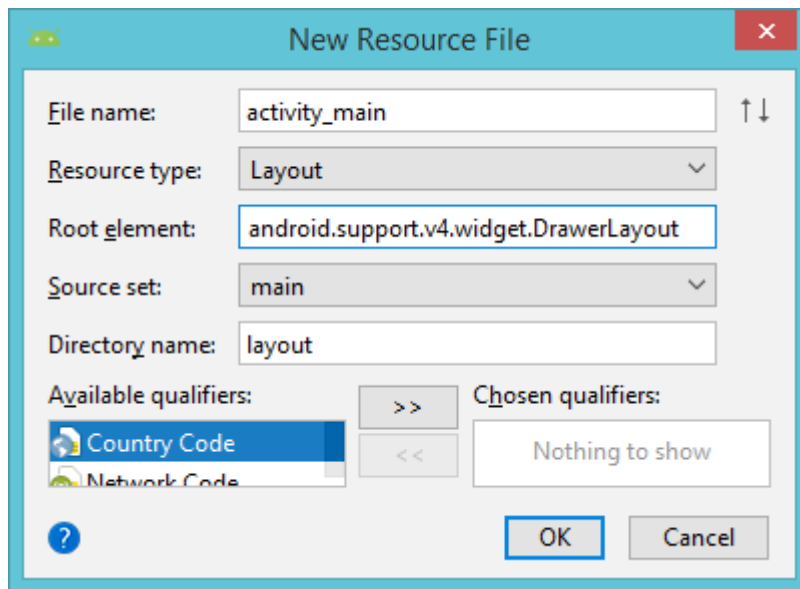
D'abord il faut vérifier que la dépendance suivante se trouve dans le fichier « **build.gradle** » du module « **app** » :

```
dependencies {  
    ...  
    implementation 'com.android.support.design:27.1.1'  
    ...  
}
```

**Remarque** : Au moment de l'ajout d'un « **Navigation Drawer** » avec le designer d'Android Studio, ce dernier vous propose d'ajouter la dépendance. Acceptez la proposition.

#### Ajouter un drawer ( tiroir) au fichier layout

Pour ajouter un « navigation drawer », il faut créer un fichier layout ayant un « **DrawerLayout** » en tant que racine :



À l'intérieur du « **DrawerLayout** », ajouter un layout pour le contenu principal de l'interface utilisateur (le layout qui s'affiche lorsque le tiroir est masqué) et une autre vue contenant le contenu du tiroir de navigation.

L'exemple suivant utilise un « **DrawerLayout** » avec deux vues enfants: un « **LinearLayout** » pour placer le contenu principal (un simple **TextView**) et un « **NavigationView** » pour le contenu du Navigation Drawer :

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true">

    <LinearLayout
        android:id="@+id/content_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <TextView
            android:id="@+id/tv_welcome"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Bienvenue" />
        </LinearLayout>

    <android.support.design.widget.NavigationView
        android:id="@+id/nav_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:fitsSystemWindows="true" />
</android.support.v4.widget.DrawerLayout>
```



Dans cet exemple :

- Le « **LinearLayout** » est configuré pour correspondre à la largeur et à la hauteur de la vue parent, car il représente toute l'interface utilisateur lorsque le tiroir de navigation est fermé. Il contient un « **TextView** » de bienvenue.
- Le « **NavigationView** » (le tiroir) doit spécifier sa gravité horizontale avec l'attribut **android:layout\_gravity**. Par défaut, le designer n'ajoute pas cet attribut. Donc il faut l'ajouter, sinon le tiroir restera toujours affiché.
- Le « **NavigationView** » utilise l'attribut « **android:fitsSystemWindows** » avec la valeur « **true** » pour s'assurer que le contenu du tiroir ne recouvre pas la barre d'état et les autres fenêtres du système. Le « **DrawerLayout** » utilise également **fitsSystemWindows** comme signe nécessaire pour insérer ses enfants (comme la vue du contenu principal), tout en affichant la barre d'état.

### Déclarer les items du menu pour le tiroir de navigation

Pour configurer les items du menu pour le « **NavigationView** », il faut indiquer une ressource de type menu avec l'attribut « **app:menu** » du « **NavigationView** », comme illustré dans l'exemple suivant :

```
<android.support.design.widget.NavigationView
    android:id="@+id/nav_view"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    android:fitsSystemWindows="true"
    app:menu="@menu/drawer_view" />
```

Créer ensuite la ressource de type menu. Exemple du fichier « **res/menu/drawer\_view.xml** » :

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_camera"
            android:icon="@drawable/ic_menu_camera"
            android:title="Import" />
        <item
            android:id="@+id/nav_gallery"
            android:icon="@drawable/ic_menu_gallery"
            android:title="Gallery" />
        <item
            android:id="@+id/nav_slideshow"
            android:icon="@drawable/ic_menu_slideshow"
            android:title="Slideshow" />
        <item
            android:id="@+id/nav_manage"
            android:icon="@drawable/ic_menu_manage"
            android:title="Tools" />
    </group>
</menu>
```

```

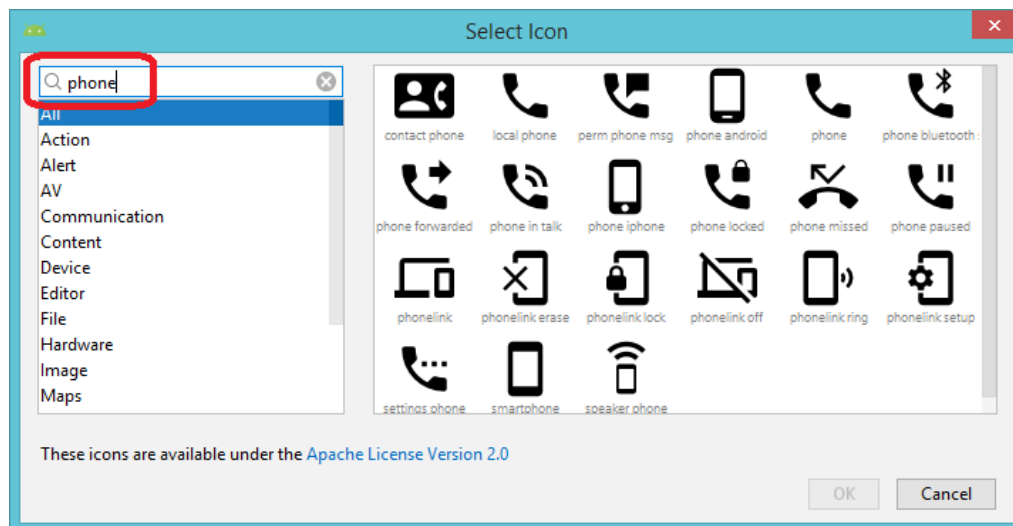
        android:id="@+id/nav_phone"
        android:icon="@drawable/ic_menu_phone"
        android:title="Phone" />
    </group>
</menu>

```

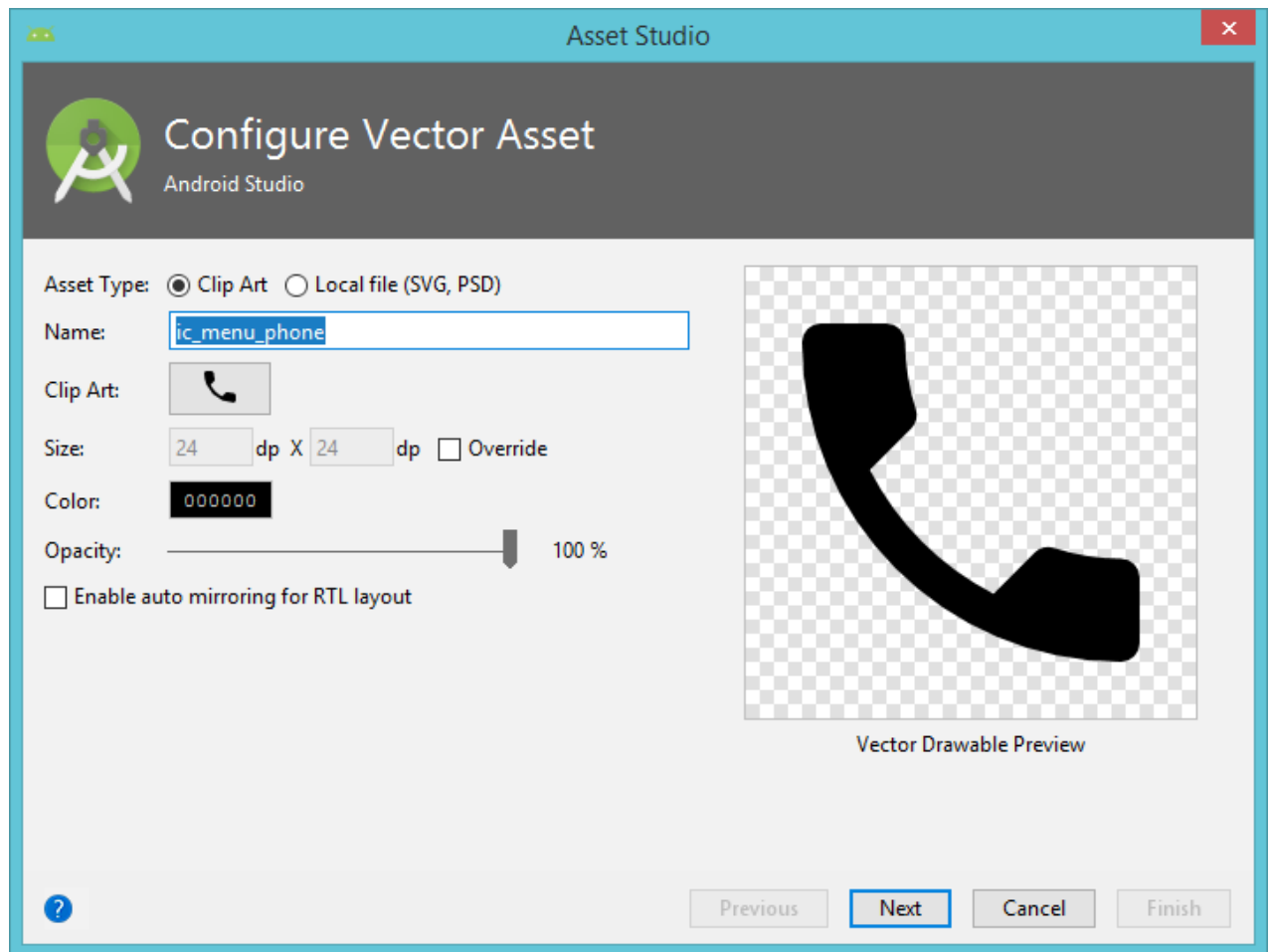
Il est possible de définir un groupe d'éléments sélectionnables en une seule fois en appliquant l'attribut « **android:checkableBehavior="single"** » au groupe. Ça permet d'indiquer quel item du menu est actuellement sélectionné.

Pour ajouter une icône, il faut suivre les étapes suivantes :

- 1) Clic droit sur le dossier « **res** » et sélectionner « **New > Vector Asset** »
- 2) Clic sur le bouton « **Clip Art** ». Sélectionner une icône à partir de la fenêtre qui s'ouvre et clic sur « **OK** ». Pour trouver une icône, il est possible de faire une recherche comme l'indique la figure suivante :



- 3) Donner un nom à l'icône choisie et appuyer sur le bouton « **Next** » (voir figure suivante) et ensuite le bouton « **Finish** ».



## Ajouter un en-tête dans le tiroir de navigation

Optionnellement, il est possible d'ajouter un en-tête en haut du tiroir en spécifiant un fichier layout avec l'attribut **app:headerLayout** comme indiqué ici:

```
<android.support.design.widget.NavigationView
    android:id="@+id/nav_view"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    android:fitsSystemWindows="true"
    app:headerLayout="@layout/nav_header"
    app:menu="@menu/drawer_view" />
```

Il faut ensuite créer l'en-tête (un fichier layout). Exemple du fichier « **res/layout/nav\_header.xml** » :

```
<?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="192dp"
    android:background="?attr/colorPrimaryDark"
    android:padding="16dp"
    android:theme="@style/ThemeOverlay.AppCompat.Dark"
    android:orientation="vertical"
    android:gravity="bottom">
```

```

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Bienvenue"
    android:textAppearance="@style/TextAppearance.AppCompat.Body1"/>
</LinearLayout>

```

## Gestion des événements de clics de navigation

Pour recevoir des rappels lorsque l'utilisateur appuie sur un item du **NavigationView**, il faut implémenter l'interface **OnNavigationItemSelectedListener** et l'associer au **NavigationView** en appelant **setNavigationItemSelectedListener()**.

Exemple :

```

NavigationView navigationView = findViewById(R.id.nav_view);
navigationView.setNavigationItemSelectedListener(
    new NavigationView.OnNavigationItemSelectedListener() {
        @Override
        public boolean onNavigationItemSelected(MenuItem item) {
            item.setChecked(true);
            int id = item.getItemId();

            if (id == R.id.nav_camera) {
                // Handle the camera action
            } else if (id == R.id.nav_gallery) {
            } else if (id == R.id.nav_slideshow) {
            } else if (id == R.id.nav_manage) {
            } else if (id == R.id.nav_phone) {
            }

            System.out.println("clic");

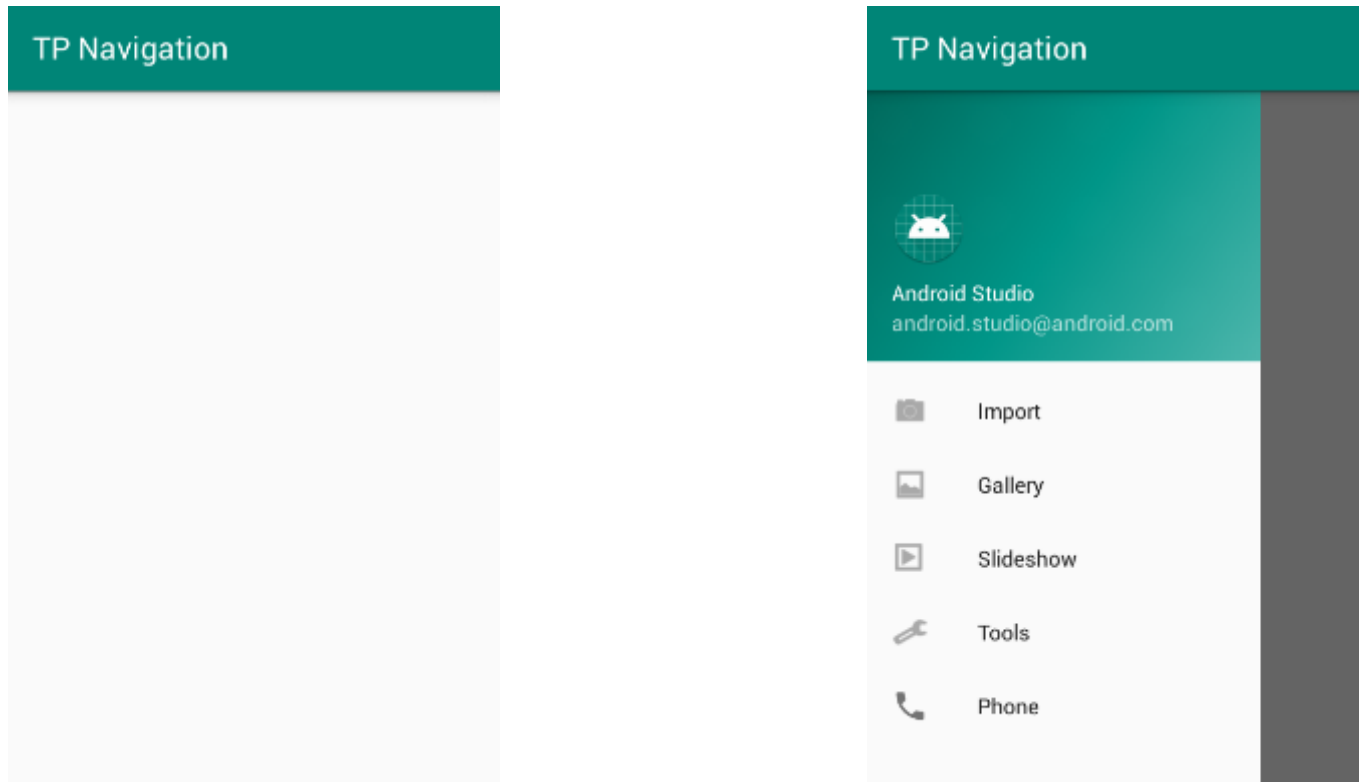
            DrawerLayout drawer = findViewById(R.id.drawer_layout);
            drawer.closeDrawer(GravityCompat.START);

            //true to display the item as selected
            return false;
        }
    });


```

Actuellement, le tiroir de navigation en état de marche, car **DrawerLayout** offre aux utilisateurs la possibilité d'ouvrir et de fermer le tiroir de navigation avec un glissement du doigt sur le côté de l'écran.

Résultat :



#### Ajouter le bouton du tiroir de navigation à la barre d'action

Il est possible d'ajouter l'icône du tiroir  en haut à gauche de la barre d'action pour permettre aux utilisateurs d'ouvrir et de fermer le tiroir en touchant cette icône. Il suffit de suivre les étapes suivantes :

- Ajouter un Toolbar au fichier layout de l'activité. Ce Toolbar prendra la place de la barre d'action par défaut de l'application. Il sera couvert par le « tiroir de navigation » (NavigationView) lorsque celui-ci est affiché. Ce Toolbar doit être placé dans le layout principale du fichier layout.

**Exemple :**

```
<LinearLayout
    android:id="@+id/content_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="?attr/colorPrimary"
        android:minHeight="?attr/actionBarSize"
        android:theme="?attr/actionBarTheme" />

    <TextView
        android:id="@+id/tv_welcome"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Bienvenue" />
</LinearLayout>
```

- Ouvrir le fichier manifeste et définir un thème de l'application sans la barre d'action, exemple :

```
android:theme="@style/AppTheme.NoActionBar"
```

- Définir ensuite ce style dans le fichier « **res/values/styles.xml** » comme suit :

```
<style name="AppTheme.NoActionBar">
    <item name="windowActionBar">false</item>
    <item name="windowNoTitle">true</item>
</style>
```

- Ajouter le Toolbar dans la barre d'action, et créer un « **ActionBarDrawerToggle** » dans l'activité :

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

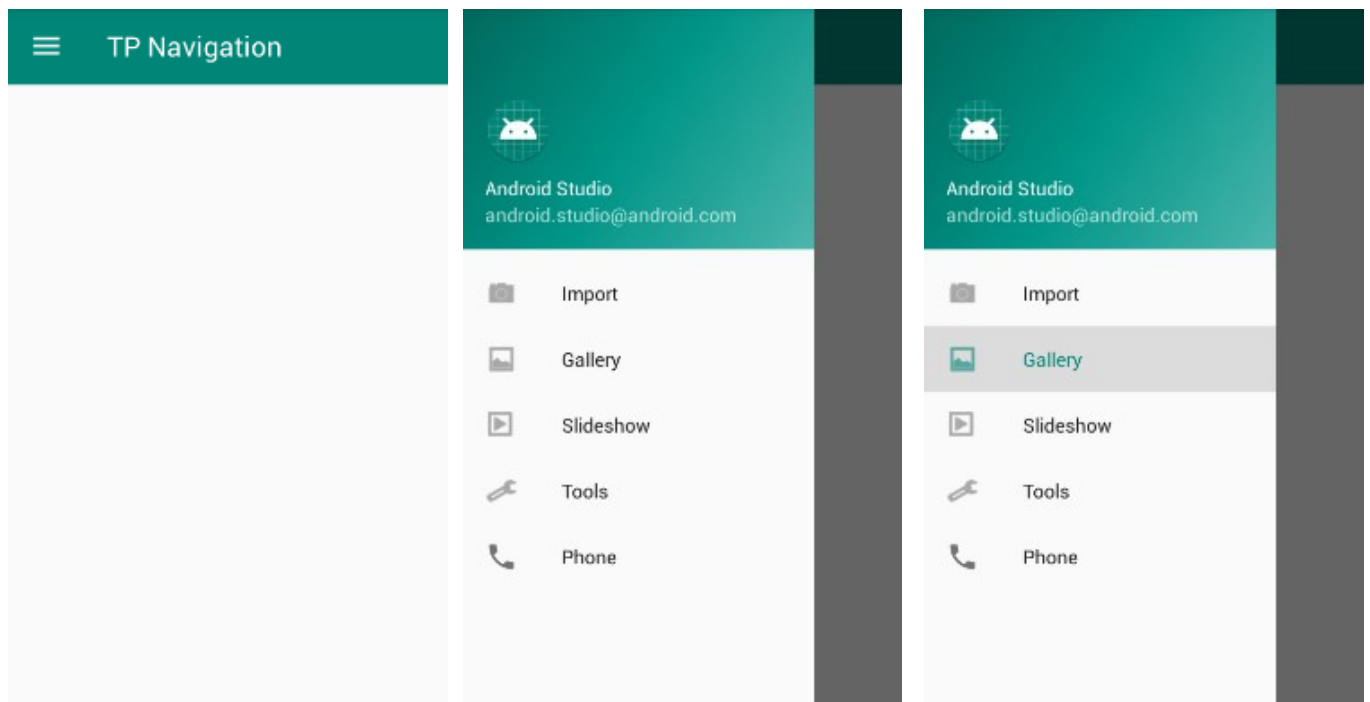
    Toolbar toolbar = findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    DrawerLayout drawer = findViewById(R.id.drawer_layout);
    ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
        this, drawer, toolbar, R.string.navigation_drawer_open,
        R.string.navigation_drawer_close);
    drawer.addDrawerListener(toggle);
    toggle.syncState();

    NavigationView navigationView = findViewById(R.id.nav_view);

    ...
}
```

## Resultat :



## 9) Les threads sous Android

Chaque application Android est gérée par un thread principal qui gère l'interface graphique et ses événements (appui sur les boutons, appui sur le menu, etc.). Ce thread est nommé `UiThread` (User Interface Thread).

Si une tâche nécessite une longue durée d'exécution, l'exécution de cette tâche par le `UiThread` bloque l'application durant toute la durée de la tâche.

**Exemple :**

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    tvCompteur = findViewById(R.id.tvCompteur);

    Button btnOk = findViewById(R.id.btnOk);
    btnOk.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            methodeLongue();
        }
    });
}

private void methodeLongue() {
    try {
        for (int i = 1; i <= 5; i++) {
            Thread.sleep(2000);
            tvCompteur.setText("" + i);
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

### Exercice : question 1 à 5

On utilise des threads secondaires pour exécuter les tâches à longue durée d'exécution (pour ne pas bloquer le `UiThread`).

**Lancement d'un thread :**

```
Thread thread = new Thread() {
    public void run() {
        methodeLongue();
    }
};
thread.start();
```



La méthode « **start()** » permet de lancer le thread. Celui-ci exécute la méthode « **run()** ».

**Remarque** : les threads secondaires n'ont pas le droit de modifier l'interface graphique, seulement le `UiThread` est autorisé à modifier cette interface.

## Exercice : question 6

Si à la fin d'exécution d'un thread on veut modifier l'interface graphique, il faut utiliser la méthode « **runOnUiThread()** » et on lui fournit un objet `Runnable` qui contient une méthode à exécuter par le `UiThread`.

**Exemple :**

```
runOnUiThread(new Runnable() {  
    @Override  
    public void run() {  
        tvCompteur.setText("Bonjour");  
    }  
});
```

**Exemple :**

```
private void methodeLongue() {  
    try {  
        for (int i = 1; i <= 5; i++) {  
            Thread.sleep(2000);  
  
            final int valeur = i;  
            runOnUiThread(new Runnable() {  
                @Override  
                public void run() {  
                    tvCompteur.setText("" + valeur);  
                }  
            });  
        }  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
}
```

## Exercice : question 7

## 10) Communication sur le réseau Internet

Remarque : Créer un compte sur le site <https://openweathermap.org/>

### a) Permission de connexion

Afin de permettre à une application Android d'accéder au réseau, il faut ajouter l'autorisation suivante dans le fichier « **AndroidManifest.xml** » :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="enim.superviseurenergie">

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        ...
        ...
        ...
```

### b) Socket

Un socket est une interface de liaison entre un client et un serveur. Le client doit créer un socket pour communiquer avec le serveur. Ensuite, il envoie ses données (destiné au serveur) au socket et il lit les réponses du serveur à partir du socket.

Le serveur est capable de communiquer avec plusieurs clients. Il attend les nouvelles connexions, et crée un socket pour chaque nouveau client. Il utilise les différents sockets pour communiquer avec tous les clients.

Il est possible de créer un socket pour l'un des deux protocoles suivants :

- **TCP**, exemple :

<https://openclassrooms.com/fr/courses/2654601-java-et-la-programmation-reseau/2668874-les-sockets-cote-serveur>

- **UDP**, exemple :

<https://openclassrooms.com/fr/courses/2654601-java-et-la-programmation-reseau/2668962-communication-reseau-avec-le-protocole-udp>

## c) Client et Serveur TCP

### Lancement du serveur

Le package « **java.net** » comprend la classe **ServerSocket**, qui permet de lancer un serveur sur un numéro de port spécifique pour écouter et accepter les connexions des clients. Ce qui nous donne :

```
int portEcoute = 12340;  
ServerSocket serveur = new ServerSocket(portEcoute);
```

Ainsi on obtient un objet de la classe **ServerSocket** sur un port spécifique. Si le numéro de port est 0, le socket est créé sur n'importe quel port libre. Dans ce cas, il faut appeler la méthode « **getLocalPort()** » de **ServerSocket** pour connaître le port utilisé.

### Côté serveur : Etablissement d'une connexion avec un nouveau client

Après le lancement du serveur, celui-ci attend les connexions des nouveaux clients. Pour chaque client qui se connecte, le serveur crée un socket (objet de la classe **Socket**). Pour attendre les nouvelles connexions et créer les sockets, on utilise la méthode « **accept()** » de **ServerSocket**.

```
Socket client = serveur.accept();
```

La méthode « **accept()** » reste bloquante tant qu'elle n'a pas détecté de connexion.

Si un nouveau client se connecte, la méthode « **accept()** » crée un socket qu'on peut utiliser pour communiquer avec le client.

### Côté client : Etablissement d'une connexion avec le serveur et création d'un socket

Le client doit également créer un socket pour communiquer avec le serveur. On utilise le constructeur de la classe « **Socket** » qui nécessite l'adresse IP du serveur et le numéro de port :

```
String adresseIpServeur = "127.0.0.1";  
int port = 12340;  
Socket server = new Socket(adresseIpServeur, port);
```

### Echange de messages

L'échange de messages entre le client et le serveur se fait en utilisant le socket créé. Il est possible de récupérer le flux d'entrée et de sortie vers la machine distante (client ou serveur). Il existe deux méthodes du socket permettant la récupération des flux :

- **getInputStream()** : permet de gérer les flux entrant (reçus).
- **getOutputStream()** : permet de gérer les flux sortant (envoyés).

Il y a plusieurs classes permettant la manipulation des flux. On s'intéresse aux deux classes suivantes :

- **BufferedReader** : permet de lire des caractères à partir d'un flux (utilisé pour lire les données reçues).
- **PrintStream** : permet d'écrire un texte dans un flux (utilisé pour envoyer des messages).

Exemple d'envoi d'un message :

```
PrintStream toRemoteMachine = new PrintStream(socket.getOutputStream());
toRemoteMachine.println("Salut, ça va ?");
```

Exemple de réception d'un message :

```
BufferedReader fromRemoteMachine =
    new BufferedReader(new InputStreamReader(socket.getInputStream()));
String message = fromRemoteMachine.readLine();
System.out.println("nouveau message reçu : " + message);
```

## Fermeture de la connexion et arrêt du serveur

A la fin de la communication entre le client et le serveur, il faut fermer le socket avec la méthode « **close()** ». Cette méthode doit être exécutée côté client et aussi côté serveur.

Lors de l'arrêt du serveur, il faut en plus fermer l'objet de type **ServerSocket** avec la méthode « **close()** » afin de libérer le port utilisé.

Exemple d'un client :

```
try {
    Socket serveur = new Socket("127.0.0.1", 12340);
    System.out.println("Connexion réussie");

    BufferedReader fromServer =
        new BufferedReader(new InputStreamReader(serveur.getInputStream()));
    PrintStream toServer = new PrintStream(serveur.getOutputStream());

    toServer.println("Salut, ça va ?");
    String reponse = fromServer.readLine();
    System.out.println("Serveur : " + reponse);

    serveur.close();
}
catch (Exception e) {
    e.printStackTrace();
}
```

## Exemple d'un serveur :

```
try {
    int portEcoule = 12340;
    ServerSocket serveur = new ServerSocket(portEcoule);
    System.out.println("Serveur lancé sur le port " + serveur.getLocalPort());

    System.out.println("Attente d'un nouveau client ...");
    Socket client = serveur.accept();

    System.out.println("Connexion du client " + client.getInetAddress().toString());
    BufferedReader fromClient =
        new BufferedReader(new InputStreamReader(client.getInputStream()));
    PrintStream toClient = new PrintStream(client.getOutputStream());

    String message = fromClient.readLine();
    System.out.println("nouveau message : " + message);
    toClient.println("OK");

    System.out.println("fin de la connexion\n");
    client.close();
    serveur.close();
}
catch (Exception e) {
    System.out.println("probleme de connexion");
}
```

## d) Client web

Le package « **java.net** » comprend la classe **URL**, qui permet de lancer un client et se connecte à une adresse url.

Il faut ensuite récupérer le flux entrant avec la méthode « **openStream** ».

La lecture du flux entrant peut se faire avec la classe **BufferedReader**. Sa methode « **readLine()** » permet de lire une seule ligne. Pour lire toutes les lignes, il faut appeler la méthode en boucle jusqu'à l'obtention d'un résultat « **null** ».

Ce qui nous donne :

```
try {
    URL url = new URL(strUrl);
    InputStream is = url.openStream();
    BufferedReader br = new BufferedReader(new InputStreamReader(is));
    String text = "";
    String line;
    while ((line = br.readLine()) != null) {
        text += line;
    }
    is.close();
    System.out.println("text : " + text);
} catch (Exception e) {
    e.printStackTrace();
}
```

## Exercice : questions 8 à 10

### Exercice :

- 1) Créer un nouveau projet vide (sans activité) avec « Android Studio » nommé « Meteo ».
- 2) Ajouter l'activité « MainActivity » : cocher la case « Launcher Activity » et décocher la case « Generate Layout File ».
- 3) Ajouter le fichier layout « **activity\_main.xml** » (utiliser LinearLayout comme racine) permettant d'afficher l'interface suivante :



- 4) Afficher le fichier layout dans MainActivity
- 5) Dans un premier temps, n'utiliser pas les threads et :
  - Ajouter la méthode « methodeLongue() » dans MainActivity. Cette méthode permet de compter de 1 à 5, d'afficher le numéro dans le TextView « Compteur » et d'attendre 2 secondes entre deux numéros successifs.
  - Ajouter une variable globale de type entier nommé « titre » initialisée à 0.
  - Ajouter un listener au bouton « OK » permettant d'appeler la méthode « methodeLongue() », d'augmenter la variable globale « titre » ( $\text{titre} += 1$ ) et d'afficher cette variable dans le titre de l'activité (`setTitle("" + titre);`)
  - Tester l'appui sur le bouton.
- 6) Modifier le listener du bouton « OK » afin d'utiliser un thread pour exécuter la méthode « methodeLongue() ». Tester à nouveau. Que se passe-t-il lorsque « methodeLongue() » tente d'afficher un numéro dans le TextView ?

- 7) Utiliser la méthode « **runOnUiThread()** » dans « **methodeLongue()** » pour afficher un numéro dans le TextView. Tester à nouveau.
- 8) Créer la méthode « **getMeteo(String ville)** » qui prend en paramètre le nom de la ville qu'on veut consulter sa météo. Cette méthode se connecte au site <https://openweathermap.org/> à travers un url de la forme suivante :

```
String strUrl = "http://api.openweathermap.org/data/2.5/weather?q=" +  
ville + "&units=metric&APPID=1a0aba74d32a8d039242a96d3f?????";
```

Cet url retourne des données JSON qu'on peut lire comme suit :

```
JSONObject json = new JSONObject(text);  
System.out.println("Ville : " + json.getString("name"));  
JSONObject mainMeteo = json.getJSONObject("main");  
System.out.println("Température C : " + mainMeteo.getString("temp"));  
System.out.println("Température C (min) : " + mainMeteo.getString("temp_min"));  
System.out.println("Température C (max) : " + mainMeteo.getString("temp_max"));  
System.out.println("Humidité : " + mainMeteo.getString("humidity"));  
JSONObject windMeteo = json.getJSONObject("wind");  
System.out.println("Vitesse du vent (m/s) : " + windMeteo.getString("speed"));
```

- 9) Modifier le listener du bouton « OK » afin d'appeler la méthode « **getMeteo(String ville)** » en lui donnant le nom de la ville saisi dans la zone de texte. Est-il possible d'appeler cette méthode à partir du thread principale ?
- 10) Compléter la méthode « **getMeteo(String ville)** » afin d'afficher les données reçues dans l'interface graphique.