

TP 1: Android

TI et IQL

S5 - 2021

Pr. Bah Slimane

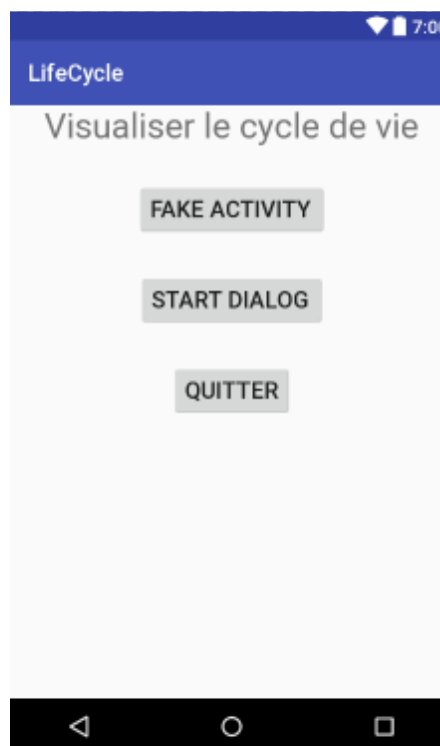
Objectif :

- Prise en main Android Studio
- Observer le cycle de vie des activités

I- Cycle de vie

L'objectif de cette partie est d'écrire une application qui nous permettra de visualiser le cycle de vie des différentes activités qui la composent.

1. Créer une activité principale en s'inspirant de l'interface suivante :



2. Ecrire le code du Bouton Quitter qui permet de terminer l'application (finish())
3. Redéfinir les différentes méthodes du cycle de vie : onStart(), onResume(), onPause(), onStop(), onDestroy() et onRestart() en ajoutant une ligne de code pour écrire sur le log les informations suivantes : le nom de l'activité et la méthode exécutée.

Pour écrire sur le log, utilisez la classe Log. Exemple :

```
Log.i("LIFECYCLE ", getLocalClassName() + " : ici onCreate");
```

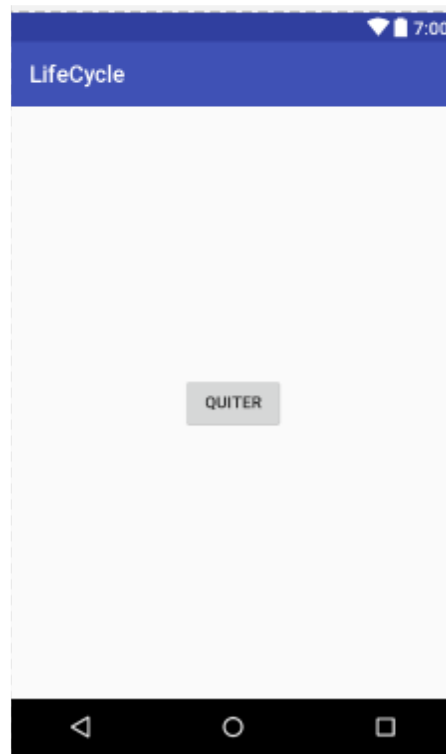
Log.i = le type du Log est information ; "LIFECYCLE" = le tag associé à ces informations

4. Exécuter l'application est observer son cycle de vie : en quittant l'application, bouton back, bouton home.
5. Dans la méthode `onDestroy()` ajouter le code java pour tuer le processus donc quitter l'application (librairie java de base : `System.exit(int)`)

Exécutez et faites pivoter l'écran de votre émulateur vers « paysage »

Oups ! On aimerait traiter ce cas, utilisez un test : `isFinishing()`

6. Ajouter une nouvelle activité (fake activity) à votre application dont l'interface est la suivante :



Ecrire le code du bouton Quitter qui termine l'activité (et non l'application)

7. Redéfinir les méthodes du cycle de vie de cette activité comme précédemment fait avec l'activité principale (Question 3)
8. Dans l'activité principale, écrire le code lié au bouton Fake Activity qui permet de démarrer cette nouvelle activité.
9. Exécuter et observer le cycle de vie des deux activités : en cliquant sur Fake Activity puis back, quitter...etc
10. Dans cette deuxième activité (Fake Activity) Ajoutez une zone de texte de saisie (`EditText`), une zone de texte de visualisation (`TextView`) et un bouton « copier » qui permet de copier le texte saisi du `EditText` vers le `TextView`. Le `TextView` doit avoir comme valeur par défaut : Hello Info

Exécutez et testez votre bouton. Juste après le teste faites pivoter l'écran « paysage ». Que se passe-t-il ?

On aimerait remédier à cela. Faites le nécessaire.

Indice : redéfinir la méthode `onSaveInstanceState()` et ajouter le code nécessaire au niveau de `onCreate()`

11. Ajouter une 3^{ème} activité avec un seul bouton Quitter qui sera appelée à partir de la 2^{ème} activité (ajouter un bouton Start seconde Fake Activity dans l'interface de la Question 5 et les traitements nécessaires pour suivre son cycle de vie, la démarrer et la terminer)
12. Exécuter et observer le cycle de vie en naviguant à travers les 3 activités.
13. Ajout d'une activité dialogue pour observer l'Etat Paused :
 - a. Ajouter une nouvelle activité : Dialog. Le layout de cette activité est donné ci-bas
 - b. Ecrire le code du bouton finishDialog (voir layout ci-bas)
 - c. Dans la méthode `onCreate()` ajouter l'instruction suivante :
`requestWindowFeature(Window.FEATURE_NO_TITLE);`
14. Dans le manifeste, repérer l'activité Dialog et ajouter la directive suivante :
`android:theme="@style/Theme.AppCompat.Dialog"`
15. Ajouter le code permettant de démarrer l'activité Dialog à partir du bouton Start Dialog de l'activité principale
16. Exécuter et observer Le cycle de vie en Cliquant sur le bouton Start Dialog

Layout de l'activité dialog (à adapter)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="225dp"
    android:layout_height="120dp"
    android:background="@color/light_blue"
    android:padding="12dp"
    tools:context="ma.emi.slim.lifecycle.DialogActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Simple Dialog"
        android:gravity="center_horizontal"
        android:textSize="24sp"
        android:textColor="@color/light_yellow"
        android:paddingBottom="12dp"
    />
```

```

<Button
    android:id="@+id/btn_finish_dialog"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="close"
    android:layout_gravity="center_horizontal"
    android:onClick="finishDialog"
/>

```

II- Plusieurs Activités :

Ecrire une activité **principale** qui permet de démarrer trois activités au choix. Un aperçu de l'activité principale est donné ici-bas

L'activité utilise 3 ImageButton (cherchez les images qui vous conviennent – iconfinder.com).

Un ImageButton est un bouton qui prend une image comme background au lieu d'un texte comme nom.



- En cliquant sur le premier bouton (Telephone), l'activité d'appel (intent implicite avec **Intent.ACTION_CALL**) est lancée en composant le numéro de la zone de texte EditText (sous forme d'URI. ex : `URI.parse("tel:0611223344")`)
- En cliquant sur le 2ème bouton (navigation) : Si l'EditText est vide, le site de l'EMI s'ouvre sinon c'est la page spécifiée par l'utilisateur dans l'EditText qui sera ouverte par le navigateur (intent implicite **Intent.ACTION_VIEW**)
- En cliquant sur le 3^{ème} bouton (ordi) une activité personnelle est démarrée : « ActivitePerso » qui affichera un bonjour. L'appel à cette activité est explicite
ActivitePerso contient un TextView avec « Welcome here » (par exemple) et un bouton « Quitter » qui permet de quitter l'activité ActivitePerso.

Ps. : Pour appeler il faudra explicitement demander la permission à l'exécution (pour Internet elle est de niveau Normal).

Pour ce faire il faut :

- a. Ajouter dans le manifest :

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

- b. Dans le code : suivre les recommandations de studio et activer le self check et y ajouter l'instruction suivante :

```
ActivityCompat.requestPermissions(this, new String[]  
{Manifest.permission.CALL_PHONE}, CALL_Perm);
```

Avec CALL_Perm est une variable de type entier de votre choix

Cette instruction permet d'afficher un message pour demander à l'utilisateur d'accorder la permission d'appeler ou pas.

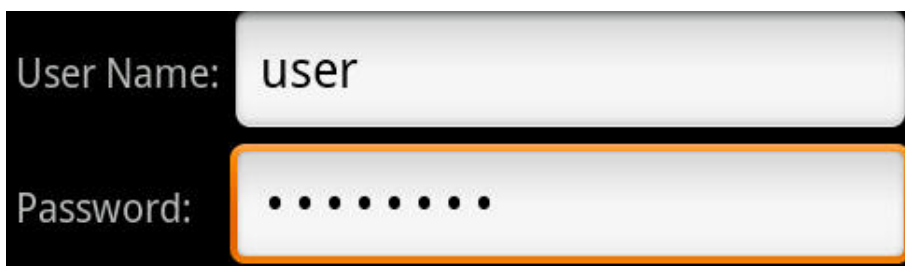
- c. Ensuite il faut redéfinir la méthode permettant de récupérer la réponse de l'utilisateur comme suit :

```
@Override  
public void onRequestPermissionsResult(int requestCode,  
String[] permissions, int[] grantResults) {  
    super.onRequestPermissionsResult(requestCode,  
permissions, grantResults);  
  
    //check the permission type using the requestCode  
    if (requestCode == CALL_Perm) {  
        //the array is empty if not granted  
        if (grantResults.length>0 &&  
grantResults[0]==PackageManager.PERMISSION_GRANTED)  
            //Toast.makeText(this, "GRANTED CALL",  
Toast.LENGTH_SHORT).show();  
    }  
}
```

Etape 2 :

- a- Modifier le comportement de l'image bouton 1 (Téléphone) de telle sorte à ce que lorsqu'on clique dessus une autre activité **Login** est lancée (on voudrait protéger les appels par mot de passe).

L'activité **Login** contient 2 TextView et deux EditText comme suit :



Et bien évidemment un bouton OK et un autre Cancel

Cancel permet de retourner à l'activité principale

OK permet de vérifier si le login et le mot de passe correspondent à TPandINFO et secret respectivement. S'ils correspondent, l'appel est effectué (l'activité principale doit envoyer le numéro à appeler à l'activité login). Sinon, pas d'appel.

Activité principale (bouton 1) : démarre login

Activité Login (bouton OK) : envoi du login et mot de passe à l'activité principale qui va vérifier avant d'effectuer l'appel ou afficher « mot de passe ou login invalide »

Activité Login (bouton Cancel) : retour à l'activité principale et afficher « opération annulée »