Cycle de vie des contrôleurs

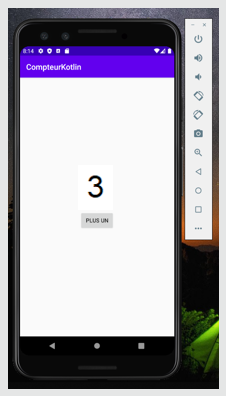
1. Objectifs

* Comprendre la problématique du changement de configuration
* Comprendre le système de pile de gestion des activités
* Comprendre le cycle de vie des activités
* Comprendre le cycle de vie des fragments
* Savoir utiliser les logs

1. Problématique lié au cycle de vie

Lorsqu'une application change de configuration, par exemple, lorsque l'on passe du mode visualisation en mode portrait au mode paysage, l'activité de l'application est détruite puis recréée, ce qui a pour conséquence de perdre l'état et les données de la page visualisée. Ci-dessous, un extrait de code ou l'on perd une information, en effet, si l'activité est recréée alors la valeur de la variable compteur sera perdu.

Rendu du code de l'IHM



Code de l'IHM

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout

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:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".MainActivity">

​

<TextView

android:id="@+id/tv\_valeur"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="0"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintRight\_toRightOf="parent"

app:layout\_constraintTop\_toTopOf="parent" />

​

<Button

android:id="@+id/btn\_compteur"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

app:layout\_constraintTop\_toBottomOf="@id/tv\_valeur"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintRight\_toRightOf="parent"

android:text="Plus un"

android:onClick="onClickCompteur"

/>

​

</androidx.constraintlayout.widget.ConstraintLayout>

Code du contrôleur

class MainActivity : AppCompatActivity() {

​

var compteur = 0;

​

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

}

​

fun onClickCompteur(view: View) {

compteur++

val tv = findViewById<TextView>(R.id.tv\_valeur)

​

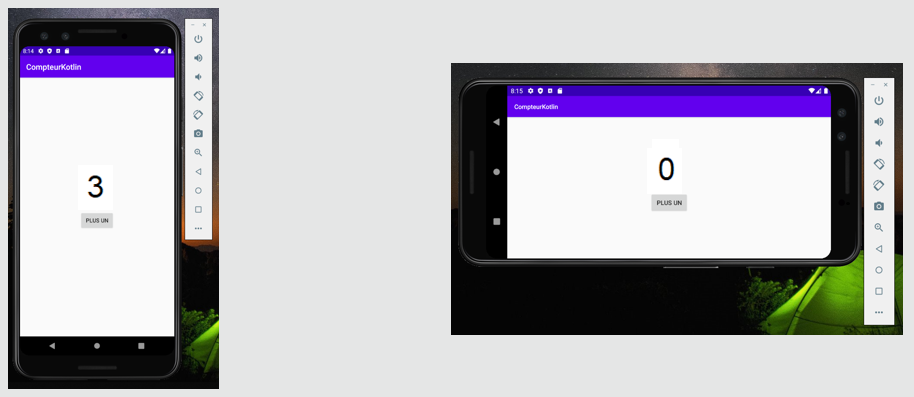
tv.text = compteur.toString()

}

}

Visualisation de la problématique

Voici, ci-dessous, un visuel permettant de comprendre comment se matérialise la perte d'information, mon compteur est égal à trois puis lorsque l'on applique une rotation de l'écran alors le compteur retourne à zéro. Ce qui est problématique.



Pour comprendre pourquoi le compteur est retourné à zéro après la rotation nous devons nous pencher sur le cycle de vie des activités. ​​​​​​​

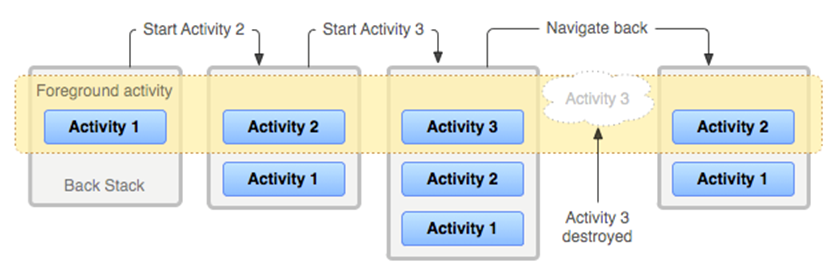
1. Les activités : La pile de gestion

Les activités sont empilées/dépilées.​​​​​​​

Une activité est empilée lorsqu’elle démarre.

Une activité est dépilée (c’est-à-dire détruite) quand on presse le bouton BACK ou par appel de la méthode finish().

Une pression sur le bouton HOME ne dépile pas l'activité. Elle passe simplement en arrière-plan.



1. Les activités : Le cycle de vie

​​​​​​​Les activités dans le système sont gérées comme une pile d'activités.

Lorsqu'une nouvelle activité est lancée, elle est placée en haut de la pile et prend le focus.

Une activité n'a que quatre états :

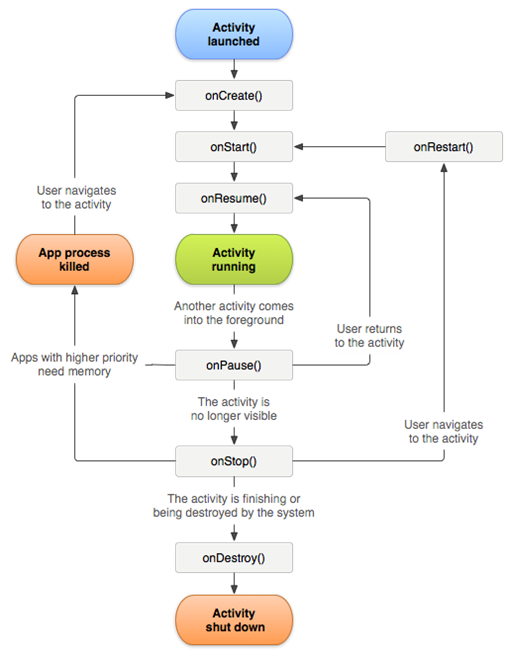
•Si une activité est au premier plan de l'écran (en haut de la pile), elle est active ou en cours d'exécution.

•Si une activité a perdu son focus mais reste visible (c'est-à-dire qu'une nouvelle activité non complète ou transparente a priorité sur votre activité), elle est mise en pause. Une activité en pause est complètement vivante (elle conserve toutes les informations d'état et de membre et reste attachée au gestionnaire de fenêtres), mais peut être détruite par le système dans des situations de mémoire extrêmement faible.

•Si une activité est complètement masquée par une autre activité, elle est arrêtée. Elle conserve toujours toutes les informations d'état et de membre, mais elle n'est cependant plus visible pour l'utilisateur.

•Si une activité est mise en pause ou arrêtée, le système peut supprimer l'activité de la mémoire en lui demandant de terminer ou en supprimant simplement son processus. Quand elle est à nouveau affichée à l'utilisateur, elle doit être complètement redémarrée et restaurée à son état précédent.

•Le diagramme suivant montre les chemins d'état importants d'une activité. Les rectangles blancs représentent les méthodes de rappel que vous pouvez implémenter pour effectuer des opérations lorsque l'activité se déplace entre les états. Les ovales colorés sont les différents états dans lesquels l'activité peut être.



Il y a trois boucles clés que vous pourriez vouloir suivre dans votre activité :

•La durée de vie totale d'une activité se produit entre le premier appel à onCreate(Bundle) et un appel final unique à onDestroy(). Une activité effectuera toutes les configurations de l'état « global » dans onCreate() et libérera toutes les ressources restantes dans onDestroy(). Par exemple, si un thread s'exécute en arrière-plan pour télécharger des données à partir du réseau, l’activité peut créer ce thread dans onCreate() puis l’arrêter dans onDestroy().

•La durée de vie visible d'une activité se produit entre un appel à onStart() et un appel correspondant à onStop(). Pendant ce temps, l'utilisateur peut voir l'activité à l'écran, bien qu’elle ne soit pas au premier plan et n'interagisse pas avec l'utilisateur. Entre ces deux méthodes, vous pouvez gérer les ressources nécessaires pour montrer l'activité à l'utilisateur. Par exemple, vous pouvez enregistrer un BroadcastReceiver dans onStart() pour surveiller les modifications qui affectent votre interface utilisateur et l'annuler dans onStop() lorsque l'utilisateur ne voit plus ce que vous affichez. Les méthodes onStart() et onStop() peuvent être appelées plusieurs fois, car l'activité devient visible et masquée pour l'utilisateur.

•La vie de premier plan d'une activité se produit entre un appel à onResume() et un appel correspondant à onPause(). Pendant ce temps, l'activité est en face de toutes les autres activités et interagit avec l'utilisateur. Une activité peut souvent passer entre les états de reprise et de pause (par exemple, lorsque l'appareil se met en veille, lorsqu'un résultat d'activité est délivré, lorsqu'une nouvelle intention est fournie), le code de ces méthodes doit donc être assez léger.

•Le cycle de vie entier d'une activité est défini par les méthodes d'activité suivantes. Tous ces éléments sont des points d'ancrage que vous pouvez remplacer pour effectuer un travail approprié lorsque l'activité change d'état. Toutes les activités implémenteront onCreate(Bundle) pour effectuer leur configuration initiale. Beaucoup vont également mettre en œuvre onPause() pour valider les modifications apportées aux données et se préparer à cesser d'interagir avec l'utilisateur. Vous devriez toujours appeler votre superclasse lors de l'implémentation de ces méthodes.

1. Démonstration de compréhension du cycle de vie

Cette démonstration permet de bien comprendre grâce aux logs le fonctionnement du cycle de vie d'une activité.