Cycle de vie

1. Objectifs

Les objectifs de ce chapitre sont les suivants :

* Présentation du cycle de vie d'une application.
* Comprendre le système de pile de gestion d'une activité.
* Comprendre le cycle de vie d'une activité.
* Comprendre le cycle de vie d'un fragment.
* Comprendre certaines problématiques liées au cycle de vie d'une application.
* Comprendre le mécanisme de sauvegarde SaveInstanceState
* Intégré les problématiques du cycle de vie afin de mieux comprendre Architecture Component.

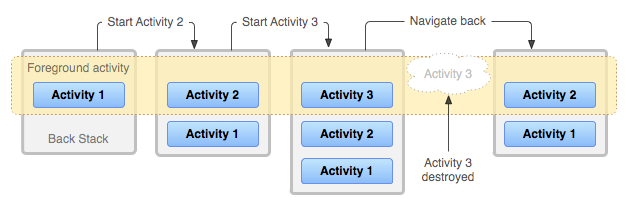
1. Présentation

Derrière l'apparition d'une nouvelle IHM à l'écran d'un appareil mobile une multitude d'actions sont exécutée en tâche de fond, ces actions permettent de construire le contenu graphique, de charger les données nécessaires à l'IHM, de libérer des ressources, de faire des sauvegardes bien d'autres choses encore. Toutes ces actions sont nécessaires au bon fonctionnement d'une application mais aussi à rendre une application plus performante.

1. Fonctionnement de la pile de gestion

La pile de gestion permet d'enregistrer les activités dans l'ordre dans lequel vous les avez ouvertes et quand vous faites un retour arrière les activités sont supprimées de cette pile de gestion. Pour exemple si je vais sur la liste de mes contacts alors l'activité 1 liste des contacts sera ajoutée puis si je clique sur un contact alors l'activité 2 du détail d'un contact sera ajouté à la pile de gestion puis si je clique sur envoyé un message alors l'activité 3 pour envoyer un message sera ajoutée à la pile de gestion puis si je fais un retour arrière alors l'activité 3 sera supprimée de la pile de gestion pour réafficher l'activité 2 du détails d'un contact puis si je fais encore un retour arrière alors l'activité 2 du détails d'un contact sera à son tour supprimée de la pile de gestion pour ensuite afficher l'activité 1 de la liste des contacts.

Voici un schéma qui vous permettra de mieux comprendre ce fonctionnement :



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.

A noter que ce fonctionnement est celui par défaut, il peut être modifié. Cette logique de fonctionnement est similaire pour les fragments.

1. Le cycle de vie d'une activité

Tout au long de leur cycle de vie, les instances d'une activité passent par différents états.

Les différents états d'une activité sont les suivants :

* En cours d'exécution
* En pause : L'activité est visible et en cours d'exécution, mais n'a pas le focus.
* Arrêtée : L'activité n'est pas visible et n'a pas le focus.
* Inexistante : L'activité n'existe pas.

La classe Activity fournit un certain nombre de fonctions appelées automatiquement (fonctions que nous appelées callback) au moment de changements d'état. Ces callbacks sont au nombre de sept :

Le callback onCreate() : appelé qu'une seule fois, au moment de la création de l'activité dans le but de créer l'ihm et d'aller chercher les données à lier à l'IHM.

Le callback onStart() : appelé à chaque fois que l'IHM de l'activité s'affiche.

Le callback onResume() : appelé à chaque fois que le focus est sur l'activité.

Le callback onPause() : appelé à chaque fois que le focus de l'activité est perdu.

Le callback onStop() : appelé à chaque fois que l'IHM de l'activité disparaît.

Le callback onDestroy() : appelé par l'activité avant sa destruction. En général, cette fonction libère les ressources.

Le callback onRestart() : appelé lorsque l'application retrouve le focus.

Le schéma ci-dessous permet d'avoir une vision globale du cycle de vie d'une activité :



Voici le code d'une activité avec tous les callbacks implémentés :

Activité avec ses sept callbacks

class MainActivity{   
   
    /\*\*   
     \* Cette fonction n'est appelée qu'une fois au moment de la   
création de l'activité.   
     \*/   
    override fun onCreate(savedInstanceState: Bundle?) {   
        super.onCreate(savedInstanceState)   
        //La fonction setContentView construit l'IHM, elle attend   
en paramètre   
        //l'identifiant du fichier xml contenant la définition de   
l'IHM de l'activité.   
        setContentView(R.layout.activity\_main)   
    }   
   
    /\*\*   
     \* Cette fonction est appelée à chaque fois que l'IHM de   
l'activité s'affiche.   
     \*/   
    override fun onStart() {   
        super.onStart()   
    }   
   
    /\*\*   
     \* Cette fonction est appelée à chaque fois que le focus est   
sur l'activité.   
     \*/   
    override fun onResume() {   
        super.onResume()   
    }   
   
    /\*\*   
     \* Cette fonction est appelée à chaque fois que le focus de   
l'activité est perdu.   
     \*/   
    override fun onPause() {   
        super.onPause()   
    }   
   
    /\*\*   
     \* Cette fonction est appelée à chaque fois que l'IHM de   
l'activité disparait.   
     \*/   
    override fun onStop() {   
        super.onStop()   
    }   
   
    /\*\*   
     \* Cette fonction est appelée lorsque l'activité était en   
arrière-plan, non visible et qu'elle redevient visible.   
     \*/   
    override fun onRestart() {   
        super.onRestart()   
    }   
   
    /\*\*   
     \* Cette fonction est la dernière fonction appelée par   
l'activité avant sa destruction.   
     \*/   
    override fun onDestroy() {   
        super.onDestroy()   
    }   
}

Cet exemple présente les différentes fonctions qui permettent de définir le comportement du cycle de vie d'une activité. Aucune instruction n'a été ajoutée dans les différentes fonctions.

1. Le cycle de vie d'un fragment

Tout au long de leur cycle de vie, les instances d'un fragment passent par différents états.

Lorsqu'un fragment est instancié, il commence dans l'état initialisé. Pour qu'un fragment traverse le reste de son cycle de vie, il doit être ajouté à un FragmentManager. Le FragmentManager est chargé de déterminer l'état des fragments.

Au-delà du cycle de vie des fragments, FragmentManager est également chargé d'attacher les fragments à leur activité hôte et de les détacher lorsque le fragment n'est plus utilisé. La classe Fragment possède deux callbacks, onAttach() et onDetach permettant de personnaliser ces actions d'attachement et de détachement.

onAttach() est donc toujours appelé avant tout changement d'état du cycle de vie.

onDetach() est donc toujours appelé après tout changement d'état du cycle de vie.

Les différents états d'un fragment sont les suivants :

* Initialisé : L'instance du fragment est créée mais le callback du fragment onCreate n'a pas encore été appelé.
* Créé :
* Démarré
* A repris
* Détruit

La vue d'un fragment à son propre cycle de vie. La figure suivante montre les états du cycle de vie du fragment et comment ils sont liés à la fois aux callbacks du cycle de vie du fragment et au cycle de vie de la vue du fragment.

fragment lifecycle states and their relation both the fragment's
            lifecycle callbacks and the fragment's view lifecycle

TODO : Expliquer les différentes relations entre cycle de vie du fragment et le cycle de vie de la vue + mettre cette image en français.

1. Changement de configuration

Lorsqu'il y a un changement de configuration du support mobile qui entraine un rechargement de l'IHM, par exemple lorsque l'écran passe du mode paysage au mode portrait, l'instance de l'activité courante est détruite puis recréée. Cela signifie que toutes les variables qui composent cette instance perdent leur valeur. Un utilisateur pourrait donc perdre tout son travail s'il avait le malheur de tourner son écran d'un côté ou d'un autre… Différentes composantes liées au cycle de vie d'une activité offrent des méthodes pour résoudre cette problématique comme par exemple le mécanisme de sauvegarde SaveInstanceState.

1. Le mécanisme de sauvegarde SaveInstanceState

Le mécanisme de sauvegarde SaveInstanceState permet de stocker les données d'une instance d'activité dans un objet juste avant que l'activité soit détruite. Quand l'activité est recréée on peut alors recharger les données à l'aide de l'objet créé au moment de sa précédente destruction.

Lorsqu'une activité s'arrête, le système appelle le callback onSaveInstanceState() permettant d'enregistrer les données de l'activité.

override fun onSaveInstanceState(sauvegarde: Bundle?) {  
    // Save the user's current game state  
    sauvegarde?.run {  
        putInt(SCORE, score)  
        putInt(NIVEAU, niveau)  
    }

//Appelle de la classe mère pour effectuer la sauvegarde  
    super.onSaveInstanceState(outState)  
}  
  
companion object {  
    val SCORE = "score\_key"  
    val NIVEAU = "niveau\_key"  
}

Lorsque votre activité est recréée après avoir été précédemment détruite, vous pouvez récupérer les données de votre instance enregistrée grâce au callback onRestoreInstanceState().

override fun onRestoreInstanceState(savedInstanceState: Bundle?) {  
    // Appelle de la classe mere pour recupérer les données précedement sauvegardées  
    super.onRestoreInstanceState(savedInstanceState)  
  
    // Chargement des données  
    savedInstanceState?.run {  
        joueurScore = getInt(SCORE)  
        joueurNiveau = getInt(NIVEAU)  
    }  
}

1. Intervention d'architecture component

Le Framework Android gère les cycles de vie des contrôleurs d'interface utilisateur, tels que les activités et les fragments. Le Framework peut décider de détruire ou de recréer un contrôleur d'interface utilisateur en réponse à certaines actions de l'utilisateur ou à des événements complètement hors de votre contrôle.

Si le système détruit ou recrée une activité, toutes les données transitoires liées à l'interface utilisateur que vous y stockez sont perdues. Par exemple, votre application peut inclure une liste d'utilisateurs dans l'une de ses activités. Lorsque l'activité est recréée pour un changement de configuration, la nouvelle activité doit récupérer la liste des utilisateurs. Pour les données simples, l'activité peut utiliser la méthode onSaveInstanceState() et restaurer ses données à partir du bundle dans onRestoreInstanceState(), mais cette approche est fastidieuse et ne convient que pour de petites quantités de données qui peuvent être sérialisées puis désérialisées, pas pour des quantités potentiellement importantes de données comme une liste d'utilisateurs ou des bitmaps.

C'est pourquoi Architecture Components, pour pallier cette limite, fournit la classe d'assistance ViewModel pour le contrôleur de l'IHM qui est responsable de la préparation des données. Les objets ViewModel sont automatiquement conservés lors des modifications de configuration afin que les données qu'ils contiennent soient immédiatement disponibles pour la prochaine activité ou l'instance de fragment.

1. Conclusion

Les activités et les fragments offrent un grand nombre de callbacks pour permettre une gestion standardisé et maitrisé du cycle de vie d'une application. Certains callback permettent de mettre en place facilement des mécanismes de sauvegarde et de chargement mais ces mécanismes conviennent aux petites applications et deviennent rapidement limités lorsque le nombre de données augmente et se complexifie, c'est pourquoi la classe ViewModel existe et permet de mettre en place un mécanisme puissant de sauvegarde et de chargement qui sera vu dans le chapitre suivant.