Polycopié de cours pour le module Applications mobiles

Chapitre III

Cycle de vie

Dans le système Android le code dans une instance d'activité est exécuté en invoquant les méthodes callback ¨appels frontaux˝ spécifiques qui correspondent à des étapes spécifiques de son cycle de vie. La classe Activity sert de point d'entrée pour une application et permet d'établir une interaction avec l'utilisateur, fournissant une fenêtre dans laquelle est dessinée l'interface utilisateur. La plupart des applications contiennent plusieurs écrans, ce qui signifie qu'ils comprennent plusieurs activités. En général, une activité d'une application est spécifiée comme activité principale, qui est le premier écran à apparaître lorsque l'utilisateur lance l'application. Chaque activité peut alors démarrer une autre activité afin d'effectuer différentes actions.

Pour utiliser les activités dans une application, devront être enregistrées dans le manifeste de l'application les informations qui les concernent. Doit être aussi géré leurs cycles de vie de manière appropriée.

III.1 Cycle de vie et appels frontaux "callback methodes"

Durant son cycle de vie, une activité passe par un certain nombre d'états. Sont associés aux différentes transitions entre les états les callbacks méthodes : `onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, `onRestart()` et `onDestroy()`

Avec les callback méthodes on peut contrôler le comportement de l'application lorsque l'utilisateur fait navigation (quitte l'application par exemple puis il l'active une autre fois).

Chaque callback méthode permet d'effectuer un travail spécifique approprié à un changement d'état donné. Faire le bon travail au bon moment et gérer les transitions correctement permet de rendre l'application plus robuste et plus performante. Une implémentation rigoureuse de ces méthodes permet d'éviter les anomalies suivantes :

- **Le crash** lorsque l'utilisateur reçoit un appel téléphonique ou passe à une autre application pendant que l'application s'exécute.
- **Consommer** (détenir) des ressources système précieuses sans les utiliser vraiment activement.
- **Crash** ou perte de la progression utilisateur lorsque l'écran transite entre l'orientation paysage et portrait.

La figure suivante (Figure III.1) résume l'essentiel du cycle de vie d'une activité :

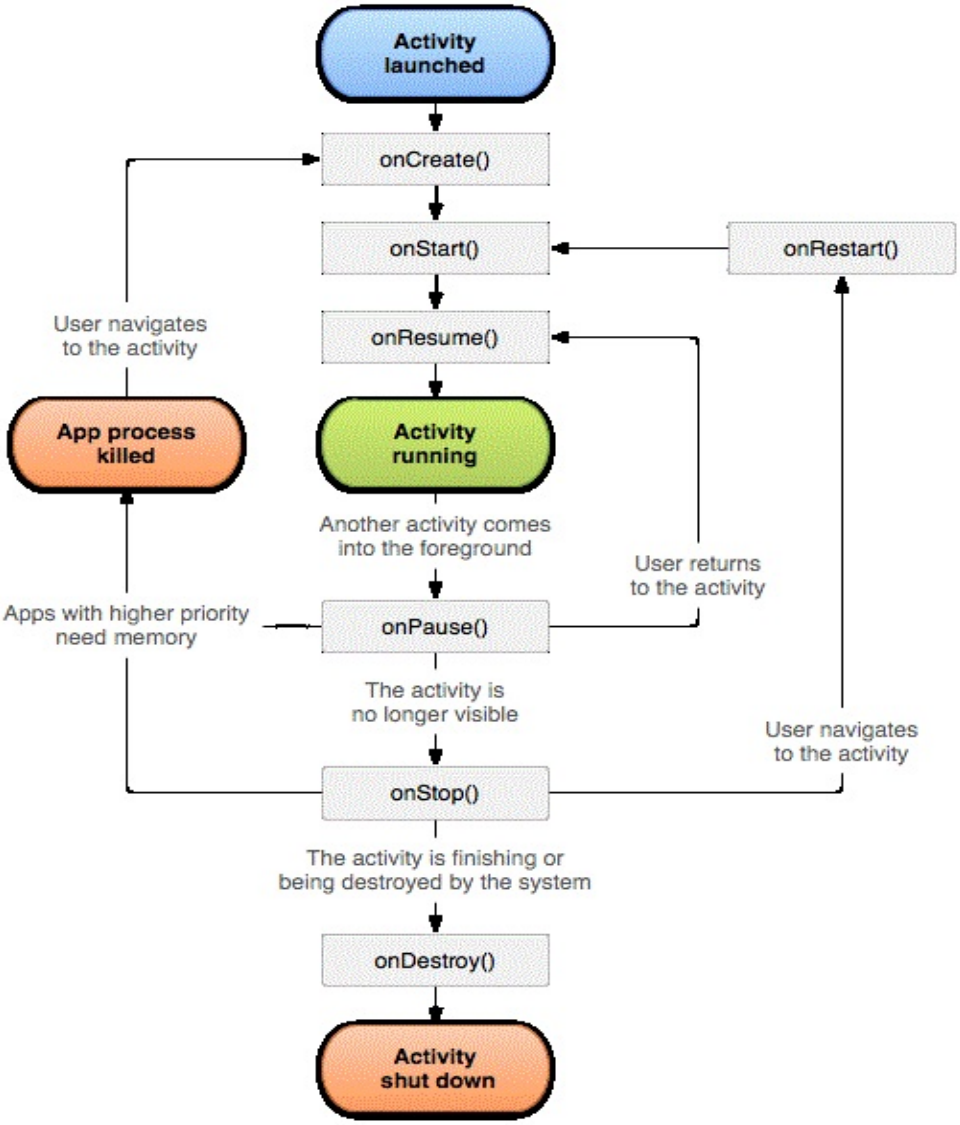

Figure III.1: "Cycle de vie d'une activité"

III.1.1 la méthode onCreate()

Dans la méthode `onCreate()` vous effectuez une logique de démarrage de base qui ne devrait se produire qu'une seule fois pour toute la durée de vie de l'activité. Par exemple, votre implémentation de `onCreate()` peut lier des données à des listes, initialiser des threads d'arrière-plan et instancier certaines variables classe. Une fois la méthode `onCreate()` terminée, l'activité entre dans l'état Started et le système appelle les méthodes `onStart()` et `onResume()` rapidement.

- exemple

```
TextView txtvw;
private String gamestate;

@Override
public void onCreate(Bundle savedInstanceState) {
    //call the super class to complete the creation of activity
    super.onCreate(savedInstanceState);
    //recovring the instance state
    if (savedInstanceState!=null) {
        gamestate = savedInstanceState.getString(Game_state_Key);
    }
    //set the user interface layout for this activity
    setContentView(R.layout.main_activity);
    //initialize TextView for later manipulation
    txtvw = (TextView)findViewById(R.id.txt_vw_id);
}

@Override
public void onRestoreInstanceState(Bundle savedInstanceState){
    txtvw.setText(savedInstanceState.getString(TEXT_VIEW_KEY));
}

//invoked when the activity may be temporarily destroyed,
//saving the instance state here.
@Override
public void onSaveInstanceState(Bundle out_state){
    out_state.putString(Game_state_Key,gamestate));
    out_state.putString(TEXT_VIEW_KEY,txtvw.getText());
}
```

La méthode `onCreate()` reçoit le paramètre **`savedInstanceState`** qui est un objet `Bundle` contenant l'état précédemment enregistré de l'activité. Si l'activité n'a jamais existé auparavant, la valeur de l'objet `Bundle` est null.

Le callback `onRestoreInstanceState()` n'est appelé que lorsqu'il existe une instance sauvegardée précédemment à l'aide de `onSaveInstanceState()` . On restaure une partie de l'état dans `onCreate()` , alors qu'on peut aussi éventuellement restaurer une autre partie d'état dans `onRestoreInstanceState()` éventuellement utilisable après la fin de `onStart()` .

III.1.2 la méthode onStart()

Lorsque l'activité entre dans l'état Started, le système invoque cette callback. L'appel `onStart()` rend l'activité visible pour l'utilisateur, car l'application prépare l'activité pour la placer au premier plan et à devenir interactive. Par exemple, cette méthode est l'endroit où l'application initialise le code qui maintient l'interface utilisateur. Il peut également enregistrer un BroadcastReceiver qui surveille les modifications qui sont reflétées dans l'interface utilisateur. La méthode `onStart()` se termine très rapidement et ne reste pas résidente dans l'état Started. Une fois cet appel terminé, l'activité passe à l'état Reprise et le système appelle la méthode `onResume()` .

III.1.3 la méthode onResume()

Lorsque l'activité entre dans l'état Reprise, elle est placée en avant plan (forground), et le système appelle le callback `onResume()`. C'est l'état dans lequel l'application interagit avec l'utilisateur. L'application reste dans cet état jusqu'à ce que quelque chose de hors application arrive à prendre le focus. Un tel événement peut s'agir par exemple, de la réception d'un appel téléphonique, la navigation d'un utilisateur vers une autre activité ou l'extinction de l'écran de l'appareil. Lorsqu'un événement d'interruption se produit, l'activité entre en état de pause et le système appelle le callback `onPause()` . Si l'activité revient à l'état Reprise à partir de l'état en pause, le système appelle une fois de plus la méthode `onResume()`. Pour cette raison, vous devez implémenter `onResume()` pour initialiser les composants que vous avez libéré pendant `onPause()`. Par exemple, vous pourrez initialiser la caméra après sa libération pendant `onPause()`.

III.1.4 la méthode onPause()

Le système appelle cette méthode autant que première indication que l'utilisateur a quitté l'activité (bien que cela ne signifie pas toujours que l'activité est détruite). Utilisez la méthode `onPause()` pour interrompre les opérations telles que les animations et la lecture de musique qui ne devraient pas continuer pendant que l'activité est en état de pause et que vous prévoyez reprendre juste un peu plus tard. Il y a plusieurs raisons pour lesquelles une activité peut entrer dans cet état. Par exemple:

- Certains événements interrompent l'exécution de l'application, comme décrit dans la section `onResume()`. C'est le cas le plus courant.
- Dans Android 7.0 (niveau API 24) ou supérieur, plusieurs applications peuvent s'exécuter en mode multi-fenêtres. Et Parce que seulement une des applications (fenêtres) qui détient le focus à un moment donné, le système met en pause toutes les autres applications.
- Avec les activités semi-transparente (telle qu'une boîte de dialogue qui s'ouvre), Tant que l'activité est encore partiellement visible sans posséder tout le focus, elle reste en pause.

Vous pouvez utiliser la méthode `onPause()` pour libérer des ressources système, telles que des BroadcastReceivers, les handlers de capteurs (comme le GPS) ou des ressources qui peuvent affecter la durée de vie de la batterie pendant que votre activité est en pause et l'utilisateur n'en a pas besoin.

`onPause()` est très bref et ne laisse pas nécessairement assez de temps pour effectuer des opérations de sauvegarde. Pour cette raison, vous ne devriez pas utiliser onPause() pour enregistrer des données d'application ou d'utilisateur, effectuer des appels réseau ou exécuter des transactions de base de données; Ce travail peut ne pas être terminé avant la fin de la méthode. Au lieu de cela, vous devriez procéder à réaliser les arrêts des lourdes taches pendant `onStop()`.

III.1.5 la méthode onStop()

Lorsque l'activité n'est plus visible pour l'utilisateur, ça indique qu'elle est passée dans un état d'arrêt et le système appelle le callback `onStop()`. Cela peut se produire, par exemple, lorsqu'une activité nouvellement lancée couvre l'ensemble de l'écran. Le système peut également appeler `onStop()` lorsque l'activité est terminée et qu'elle est sur le point d'être terminée.

Dans la méthode `onStop()`, l'application devrait libérer presque toutes les ressources qui deviennent pas vraiment nécessaires alors que l'utilisateur ne les utilise pas à ce moment. Par exemple, si vous avez enregistré un BroadcastReceiver dans `onStart()` pour intercepter les modifications susceptibles d'affecter votre interface utilisateur, vous pouvez annuler son enregistrement dans `onStop()`, car l'utilisateur ne peut plus l'afficher. Il est également important que vous utilisiez `onStop()` pour libérer des ressources qui risquent de puiser la mémoire, car il est possible pour le système de tuer le processus d'hébergement de votre activité sans appeler le callback final de l'activité `onDestroy()`.

III.1.6 la méthode onDestroy()

Lorsque l'activité n'est plus visible pour l'utilisateur, ça indique qu'elle est passée dans un état d'arrêt et le système appelle le callback `onStop()`. Cela peut se produire, par exemple, lorsqu'une activité nouvellement lancée couvre l'ensemble de l'écran. Le système peut également appeler `onStop()` lorsque l'activité est terminée et qu'elle est sur le point d'être terminée.

Dans la méthode `onStop()`, l'application devrait libérer presque toutes les ressources qui deviennent pas vraiment nécessaires alors que l'utilisateur ne les utilise pas à ce moment. Par exemple, si vous avez enregistré un BroadcastReceiver dans `onStart()` pour intercepter les modifications susceptibles d'affecter votre interface utilisateur, vous pouvez annuler son enregistrement dans `onStop()`, car l'utilisateur ne peut plus l'afficher. Il est également important que vous utilisiez `onStop()` pour libérer des ressources qui risquent de puiser la mémoire, car il est possible pour le système de tuer le processus d'hébergement de votre activité sans appeler le callback final de l'activité `onDestroy()`.