Fondamentaux d'Android

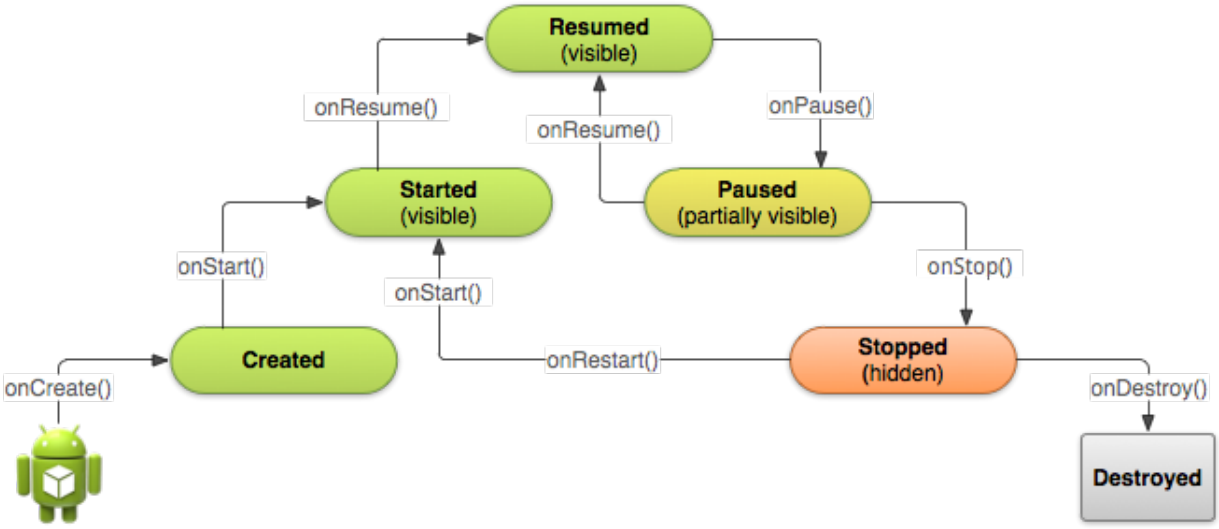
5. Cycle de vie et état de l'activité

Hatem Aziza - October 2024

Developers

Introduction

Dans cet atelier, vous en apprendrez davantage sur le cycle de vie de l'activité. Le cycle de vie est l'ensemble des états dans lesquels une activité peut se trouver tout au long de sa durée de vie, du moment où elle est créée jusqu'au moment où elle est détruite et où le système récupère ses ressources. Lorsqu'un utilisateur navigue entre les activités de votre application (ainsi que vers et hors de votre application), les activités passent d'un état à l'autre au cours de leur cycle de vie.



Chaque étape du cycle de vie d'une activité correspond à une méthode de rappel : **onCreate()**, **onStart()**, **onPause()**, etc. Lorsqu'une activité change d'état, la méthode de rappel associée est invoquée. Vous avez déjà vu l'une de ces méthodes : onCreate(). En remplaçant l'une des méthodes de rappel du cycle de vie dans vos classes Activity, vous pouvez modifier le comportement par défaut de l'activité en réponse aux actions de l'utilisateur ou du système.

Ce que vous apprendrez

- Comment fonctionne le cycle de vie d'une activité.
- Quand l'activité démarre, s'arrête, s'arrête et est détruit.
- Effet des actions (telles que les modifications de configuration)
- Comment conserver l'état d'une activité tout au long des événements du cycle de vie.

Aperçu de l'application

Dans cette atelier, l'apparence et le comportement de l'application sont à peu près identiques à ceux du dernier atelier de programmation **TwoActivities**. Il contient deux activités et donne à l'utilisateur la possibilité d'envoyer entre elles. Les modifications que vous apportez à l'application dans cet atelier n'affecteront pas son comportement utilisateur visible.

Tâche 1 : ajouter des rappels de cycle de vie à TwoActivities

Changez le groupe de vues racine en LinearLayout

Dans cette tâche, vous implémenterez toutes les **méthodes de rappel** du **cycle de vie** d'une activité pour imprimer des messages sur **logcat** lorsque ces méthodes sont invoquées.
Ces messages de journal vous permettront de voir quand le cycle de vie change d'état et comment ces changements d'état du cycle de vie affectent votre application pendant son exécution.

Copiez le projet TwoActivities

Pour les tâches de cet atelier, Copiez le projet **TwoActivities** existant que vous avez créé lors du dernier TP.

Implémenter les rappels dans MainActivity

1. Ouvrez le projet **TwoActivities** dans Android Studio et ouvrez **MainActivity**

2. Dans la méthode onCreate(), ajoutez les instructions suivantes :
   ```
   Log.d(LOG_TAG, "-------");
   Log.d(LOG_TAG, "onCreate");
   ```

3. Ajoutez la méthode de rappel onStart(), avec une instruction dans le journal pour cet événement :

   ```
   @Override
   public void onStart(){
      super.onStart();
      Log.d(LOG_TAG, "onStart");
   }
   ```

4. Utilisez la méthode onStart() comme modèle pour implémenter les méthodes rappels onPause(), onRestart(), onResume(), onStop() et onDestroy(). *Toutes les méthodes de rappel ont les mêmes signatures (sauf le nom).*

5. Exécutez votre application.

Cliquez sur l'onglet **Logcat** en bas d'Android Studio pour afficher le volet **Logcat**. Vous devriez voir trois messages de journal indiquant les trois états du cycle de vie de l'activité par lesquels la transition a commencé :

D/MainActivity: -------
D/MainActivity: onCreate
D/MainActivity: onStart
D/MainActivity: onResume

Implémenter des rappels de cycle de vie dans SecondActivity

Maintenant que vous avez implémenté les méthodes de rappel du cycle de vie pour MainActivity, faites de même pour SecondActivity.

1. Ouvrez **SecondActivity**. En haut de la classe, ajoutez une constante pour la LOG_TAG:

   private static final String LOG_TAG = SecondActivity.class.getSimpleName();

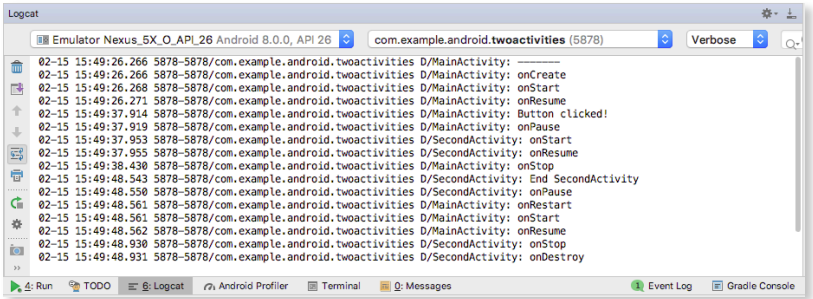
2. Ajoutez les rappels du cycle de vie et les instructions de journalisation au second Activity. (Vous pouvez copier et coller les méthodes de rappel à partir de MainActivity.)

3. Ajoutez une instruction de journal à la méthode **returnReply()** juste avant la méthode **finish()**:

   Log.d(LOG_TAG, "End SecondActivity");

Observez le journal pendant l'exécution de l'application

1. Exécutez votre application.
2. Cliquez sur l' onglet **Logcat** en bas d'Android Studio pour afficher le volet **Logcat** .
3. Saisissez Activité dans la zone de recherche. Le logcat Android peut être très long et encombré. Étant donné que la constante LOG_TAG de chaque classe contient les mots **MainActivity** ou **SecondActivity**, ce mot-clé vous permet de filtrer le journal uniquement pour les éléments qui vous intéressent.



4. Expérimentez en utilisant votre application et notez les événements du cycle de vie qui se produisent en réponse à différentes actions. En particulier, essayez ces choses :
   A) Utilisez l'application normalement (envoyez un message, répondez avec un autre message).
   B) Utilisez le bouton Retour pour revenir du deuxième Activity à l'activité principale.
   C) Utilisez la flèche vers le haut dans la barre d'applications pour revenir du deuxième Activity à l'activité principale.
   D) Faites pivoter l'appareil sur la principale et la seconde activité à différents moments dans votre application et observez ce qui se passe dans le journal et sur l'écran.
   E) Appuyez sur le bouton d'aperçu (le bouton carré à droite de Accueil) et fermez l'application (appuyez sur le X).
   F) Revenez à l'écran d'accueil et redémarrez votre application.

CONSEIL : Si vous exécutez votre application dans un émulateur, vous pouvez simuler la rotation avec **Control+L** ou **Control+R**.

Code de solution de la tâche 1

Les extraits de code suivants montrent le code de solution pour la première tâche.

Activité principale

Les extraits de code suivants montrent le code ajouté dans **MainActivity**.

La méthode onCreate():

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

    // Log the start of the onCreate() method.
    Log.d(LOG_TAG, "-------");
    Log.d(LOG_TAG, "onCreate");

    // Initialize all the view variables.
    mMessageEditText = findViewById(R.id.editText_main);
    mReplyHeadTextView = findViewById(R.id.text_header_reply);
    mReplyTextView = findViewById(R.id.text_message_reply);
}
```

Les autres méthodes de cycle de vie :

```
@Override
```

```
protected void onStart() {
    super.onStart();
    Log.d(LOG_TAG, "onStart");
}

@Override
protected void onPause() {
    super.onPause();
    Log.d(LOG_TAG, "onPause");
}

@Override
protected void onRestart() {
    super.onRestart();
    Log.d(LOG_TAG, "onRestart");
}

@Override
protected void onResume() {
    super.onResume();
    Log.d(LOG_TAG, "onResume");
}

@Override
protected void onStop() {
    super.onStop();
    Log.d(LOG_TAG, "onStop");
}

@Override
protected void onDestroy() {
    super.onDestroy();
    Log.d(LOG_TAG, "onDestroy");
}
```

Deuxième activité

Les extraits de code suivants montrent le code ajouté dans **SecondActivity**.

En tête de classe **SecondActivity**:

```
private static final String LOG_TAG = SecondActivity.class.getSimpleName();
```

La méthode **returnReply()** :

```
public void returnReply(View view) {
    String reply = mReply.getText().toString();
    Intent replyIntent = new Intent();
    replyIntent.putExtra(EXTRA_REPLY, reply);
    setResult(RESULT_OK, replyIntent);
    Log.d(LOG_TAG, "End SecondActivity");
    finish();
}
```

Les **autres méthodes** de cycle de vie : *Identique à MainActivityci-dessus.*

Tâche 2 : Enregistrer et restaurer l'état de l'instance d'activité

En fonction des ressources système et du comportement des utilisateurs, chaque élément de l'activité de votre application peut être détruit et reconstruit beaucoup plus fréquemment que vous ne le pensez.
Vous avez peut-être remarqué ce comportement dans la dernière section lorsque vous avez fait pivoter l'appareil ou l'émulateur. La rotation de l'appareil est un exemple de modification de la configuration d'un appareil. Bien que la rotation soit la plus courante, tous les changements de configuration entraînent la destruction de l'activité et sa recréation comme s'elle était nouvelle.
Si vous ne tenez pas compte de ce comportement dans votre code, lorsqu'un changement de configuration se produit, le layout de votre activité peut revenir à son apparence par défaut et à ses valeurs initiales, et vos utilisateurs peuvent perdre leurs données ou l'état de leur progression.

L'**état de chaque activité** est stocké sous la forme d'un ensemble de **paires clé/valeur** dans un objet **Bundle** appelé état d'instance de l'activité. Le système enregistre les informations d'état par défaut dans l'état de l'instance Bundle juste avant l'arrêt de l'instance de l'activité et les transmet à la nouvelle instance à restaurer.

Pour éviter de perdre des données lorsqu'elles sont détruites, vous devez implémenter la méthode **onSaveInstanceState()**. Le système appelle cette méthode sur votre Activité (entre onPause()et onStop()) lorsque l'activité est soit détruite soit recréée.

Les données que vous enregistrez dans l'état de l'instance sont spécifiques à cette instance uniquement pendant la session d'application en cours. Lorsque vous arrêtez et redémarrez une nouvelle session d'application, l'état de l'instance est perdu et revient à son apparence par défaut. Si vous devez enregistrer les données utilisateur entre les sessions de l'application, utilisez les préférences partagées ou une base de données.

Enregistrez l'état de l'instance d'activité avec onSaveInstanceState( )

Les informations d'état de certains éléments View de votre layout sont automatiquement enregistrées lors des modifications de configuration, et la valeur actuelle d'EditText est l'un de ces cas.
Dans cette tâche, vous ajoutez du code pour préserver l'état d'instance des deux éléments TextView à l'aide de **onSaveInstanceState()**.

1. Ouvrez MainActivity .
2. Ajoutez cette implémentation de squelette de **onSaveInstanceState()**,

```
@Override
public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
}
```

3. Vérifiez si l'en-tête est actuellement visible, et si c'est le cas, mettez cet état de visibilité dans l'état Bundle avec la méthode putBoolean() et la clé "reply_visible".

```
if (mReplyHeadTextView.getVisibility() == View.VISIBLE) {
   outState.putBoolean("reply_visible", true);
}
```

4. Dans cette même vérification, ajoutez le texte de réponse dans le fichier Bundle.

```
outState.putString("reply_text",mReplyTextView.getText().toString());
```

Restaurez l'état de l'instance d'activité dans onCreate( )

Une fois que vous avez enregistré l'état de l'instance de l'activité, vous devez également le restaurer lors de sa recréation. Vous pouvez le faire soit dans **onCreate()**, soit en implémentant la méthode **onRestoreInstanceState()**, qui est appelé après **onStart()** après la création de l'activité.
La plupart du temps, le meilleur endroit pour restaurer l'état de l'activité est en **onCreate()**, afin de garantir que l'état soit disponible dès que possible. Il est parfois pratique de le faire dans **onRestoreInstanceState()**.

1. Dans la méthode **onCreate()**, une fois les variables initialisées avec findViewById(), ajoutez un test pour vous assurer que savedInstanceState n'est pas nul.

```
// Initialize all the view variables.
mMessageEditText = findViewById(R.id.editText_main);
mReplyHeadTextView = findViewById(R.id.text_header_reply);
mReplyTextView = findViewById(R.id.text_message_reply);
```

```
// Restore the state.
if (savedInstanceState != null) {
}
```

Lorsque votre activité est créé, le système transmet l'état **Bundle** comme seul argument de **onCreate()**. La première fois qu'elle est appelée et que votre application démarre, c'est le Bundle est null car il n'y a aucun état existant la première fois que votre application démarre.

2. À l'intérieur de cette vérification, obtenez la visibilité actuelle (vrai ou faux) Bundle avec la clé "reply_visible".

```
if (savedInstanceState != null) {
   boolean isVisible =
            savedInstanceState.getBoolean("reply_visible");
}
```

3. Ajoutez un test sous cette ligne précédente pour la variable isVisible.

```
if (isVisible) {
}
```

S'il y a une clé **reply_visible** dans l'état **Bundle** (et isVisible est true), vous devrez restaurer l'état.

4. À l'intérieur ddu test **isVisible**, rendez l'en-tête visible.

```
mReplyHeadTextView.setVisibility(View.VISIBLE);
```

5. Obtenez le message de réponse texte du Bundle avec la clé "reply_text"et définissez la réponse TextView pour afficher cette chaîne.

```
mReplyTextView.setText(savedInstanceState.getString("reply_text"));
```

6. Rendez également la réponse TextView visible :

```
mReplyTextView.setVisibility(View.VISIBLE);
```

7. Exécutez l'application. Essayez de faire pivoter l'appareil ou l'émulateur pour vous assurer que le message de réponse (le cas échéant) reste à l'écran après la recréation de l'activité.

Code de solution de la tâche 2

Les extraits de code suivants montrent le code de solution pour cette tâche.

Activité principale

Les extraits de code suivants montrent le code ajouté dans MainActivity.

```
        @Override
        public void onSaveInstanceState(Bundle outState) {
          super.onSaveInstanceState(outState);
          // If the heading is visible, message needs to be saved.
          // Otherwise we're still using default layout.
          if (mReplyHeadTextView.getVisibility() == View.VISIBLE) {
             outState.putBoolean("reply_visible", true);
             outState.putString("reply_text",
                        mReplyTextView.getText().toString());
          }
        }

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

  Log.d(LOG_TAG, "-------");
  Log.d(LOG_TAG, "onCreate");

  // Initialize all the view variables.
  mMessageEditText = findViewById(R.id.editText_main);
  mReplyHeadTextView = findViewById(R.id.text_header_reply);
  mReplyTextView = findViewById(R.id.text_message_reply);

  // Restore the saved state.
  // See onSaveInstanceState() for what gets saved.
  if (savedInstanceState != null) {
     boolean isVisible =
                savedInstanceState.getBoolean("reply_visible");
     // Show both the header and the message views. If isVisible is
     // false or missing from the bundle, use the default layout.
     if (isVisible) {
        mReplyHeadTextView.setVisibility(View.VISIBLE);
        mReplyTextView.setText(savedInstanceState
                        .getString("reply_text"));
        mReplyTextView.setVisibility(View.VISIBLE);
     }
  }
}
```