Développement mobile : Android

Lesson 2

Carl Esswein

Dernière mise à jour : V2.2 - février 2024.

Table des matières

Objectifs

L'objectif de cette deuxième séance est de vous présenter comment étoffer vos applications, d'abord en prenant en charge les actions utilisateurs par le biais de **listeners** puis – et surtout – en abordant le **concept important** d'**Intent** qui va permettre à vos composants Android de dialoguer entre eux, et avec les composants d'autres applications ! Vous verrez ensuite comment gérer les **permissions** dans vos applications android.
Cette séance sera également l'occasion de manipuler quelques nouveaux éléments d'IHM tels que les Spinner ou les menus.

1 Les listeners

Nous avons terminé la séance précédente en construisant l'IHM ainsi que la partie *métier* du code d'une application « Plus ou moins » (Lesson 1, Q7) ; nous allons maintenant la rendre utilisable en rendant le clic sur le bouton actif.

Tout comme le modèle événementiel de Swing permettait de réagir à des événements utilisateur dans un programme java, android fournit la possibilité d'attacher des listeners aux différents composants (`View`) de vos IHM.

1.1 How-to

Q1. Modifiez votre application « Plus ou moins » ainsi :

1.1. Changez la signature de votre méthode `check()` pour qu'elle prenne un paramètre de type `View`.
1.2. Ajoutez l'attribut `android:onclick="check"` au bouton dans votre layout XML puis testez.

Pour les événements de type « clic », très courants, cela peut se faire directement dans le fichier de layout, comme ci-dessus, ou bien dans le code java en implémentant un listener de type **`OnClickListener`**. Cette interface est définie comme une interface interne de la classe `View` et ne définit qu'une méthode :

```
public void onClick(View view)
```

Q2.

2.1. Supprimez l'attribut `android:onclick` de la question 1.
2.2. **Modifiez** votre activité pour qu'elle implémente `View.OnClickListener` et que la méthode `onClick(View view)` appelle votre méthode `check(View view)`.
2.3. **Testez**. Que manque-t-il ?
2.4. Si nécessaire, complétez votre application pour la rendre totalement fonctionnelle.

Q3. Une fois le nombre trouvé par l'utilisateur, proposez-lui de recommencer une nouvelle partie ou de quitter. Ajoutez les `View` et les listeners nécessaires et testez.

Vous pouvez explicitement mettre fin à une activité en appelant sa méthode `finish()`.

Il existe d'autres types d'événements auxquels vous pouvez faire réagir vos IHM ; certains sont associés à des listeners définis en tant qu'interface interne de `View` (*Cf.* liste complète sur http://developer.android.com/reference/android/view/View#nestedclasses), d'autres à des interfaces internes à d'autres composants. A ma connaissance il n'y a qu'*OnClick* qui soit accessible via un attribut XML dans le fichier de layout.

1.2 Exercice

Q4. Créez un nouveau projet à partir d'une *Empty View Activity*, puis :

4.1. modifiez le layout de cette activité pour qu'il affiche une question et une liste déroulante (`Spinner`[1]) comme ci-contre.

(le background, c'est optionnel…)

Remplissez votre `Spinner`, via un `ArrayAdapter<CharSequence>`, à partir d'une ressource de type tableau de `String` créée dans *strings.xml*.

Pour cela vous créerez un adapter à l'aide de la méthode `createFromResource()`.

L'`Adapter` sert de « pont » pour faire le lien entre les données et la vue (votre `Spinner`). *Cf.* https://developer.android.com/develop/ui/views/components/spinner

4.2. Ajoutez un **listener** à votre `Spinner` : faites apparaître un simple `Toast` qui affiche la réponse sélectionnée par l'utilisateur.

4.3. A partir du code java, modifiez votre gestion de l'adapteur pour que vous puissiez ajouter/supprimer des éléments dans le `Spinner` par rapport à l'initialisation obtenue via la ressource. Testez par exemple en rajoutant « another planet ».

Ne construisez plus ici votre `Adapter` via une ressource directement, mais via un attribut – à définir – que vous initialiserez avec la ressource (Utilisez `getResources().get…` pour accéder aux ressources à partir du code java). Ainsi l'utilisation de la méthode `Adapter.add(…)` sera autorisée et ne lèvera pas de `UnSupportedOperationException`.

`getPreferences(Context.MODE_PRIVATE)`, combiné avec – par exemple – des méthodes telles que `putString(String, String)` et `getString(String, String)` vous permettent d'enregistrer et de récupérer des couples clé-valeur dans vos applications android. C'est le système des `SharedPreferences`[2]. Ces « préférences partagées » survivent à la fermeture de votre application.
`getPreferences(...)` vous donne accès au fichier de préférences par défaut de votre activité. Vous pouvez alternativement définir plusieurs fichiers différents identifiés par des noms (`String`).

Attention, la lecture (`getXxx(…)`) est directe tandis que l'écriture doit passer par l'utilisation d'un éditeur et met en œuvre des transactions (`commit()` ou `apply()` requis *etc.*).

4.4. Utilisez les *SharedPreferences* pour enregistrer la réponse sélectionnée par l'utilisateur avant la fermeture de l'application (par exemple via le listener de sélection, *cf.* Q4.2), et restaurez cette valeur au lancement de l'appli.

[1] http://developer.android.com/reference/android/widget/Spinner
[2] http://developer.android.com/training/data-storage/shared-preferences

2 Lancement d'une activité : le concept d'*Intent*

Nous allons maintenant aborder la notion d'`Intent` qui est un concept central qui différencie nettement le modèle android des modèles concurrents en développement mobile. Commençons par utiliser un Intent pour lancer une seconde activité à partir d'une première.

2.1 Premier pas : expicitement

2.1.1 Petit détour par les menus, les fragments, et le *view binding*

Q5. Créez un nouveau projet à partir d'une *Basic Views Activity*. C'est l'occasion de manipuler rapidement quelques nouveaux éléments d'IHM Android courants. **Lancez** l'app pour visualiser son rendu actuel.

Observez le contenu du projet. Vous verrez qu'Android Studio a créé trois fichiers Java (une activité et deux fragments), quatre layouts (un pour chaque fragment et deux pour l'activité) ainsi que deux ressources XML importantes : *menu_main.xml*, dans */res/menu* et *nav_graph.xml* dans */res/navigation*.
L'activité principale est bien sûr toujours `MainActivity`. Elle est référencée dans *AndroidManifest.xml* comme activité principale de l'application. Le layout *activity_main.xml* ne prend pas directement en charge le contenu "métier" présenté à l'utilisateur (celui-ci est géré via un *include* du layout *content_main.xml*) ; en revanche, il fournit la prise en charge de la `Toolbar` (la bande bleue en haut, incluant le menu) et du `FloatingActionButton` (en bas à droite).

5.1. Ajoutez un élément "To Activity2" dans le menu, et testez-le en affichant un simple `Toast` pour l'instant.

Ce wizard de projet *Basic Views Activity* met en œuvre les `Fragment` qui permettent de développer des IHM de façon plus modulaire. En deux mots, *content_main.xml* affiche le fragment de départ. La suite est déterminée par les actions de l'utilisateur. Les transitions de navigation entre les différents fragments sont définies dans *nav_graph.xml*. Les listeners d'événement (*touch* sur un bouton ou un menu par exemple) sont gérés comme d'habitude (*cf.* `onClick` *etc.* que nous avons déjà vu) ; ils utilisent la méthode `navigate` pour faire référence à une action définie dans *nav_graph.xml*.
Nous n'allons cependant pas utiliser les *fragment* dans la suite de ce paragraphe car celui-ci est orienté *Intent*.

Notez enfin que dans la méthode `onCreate` de `MainActivity` l'utilisation de `findViewById` a été remplacée par l'utilisation du *view binding*. En deux mots, le compilateur génère une classe java `ActivityMainBinding` qui contient un attribut par view référencée dans le layout de l'activité (un attribut `toolbar` et un attribut `fab` dans notre cas). *Cf.* https://developer.android.com/topic/libraries/view-binding#activities pour plus de détail.

5.2. Faites simplement le changement nécessaire pour que le 2e fragment soit affiché en premier lors du lancement de l'activité.

Revenons maintenant à cette notion d'Intent.

2.1.2 Intent explicite

Q6. Créez une seconde activité, par exemple `Activity2`, à partir du wizard *Empty Views Activity* et faites en sorte qu'elle affiche « I dit it ! » au centre de l'écran.

6.1. Dans la méthode dans laquelle vous faisiez le `Toast` dans la Q5.1, créez, en vous aidant de la doc d'`Intent`[3] une instance `myIntent` d'`Intent` à partir du contexte courant (`this`) et de la classe `Activity2.class`.

Ceci est valide car `Activity` hérite de `Context`. De manière générale, à partir d'un autre type de composant (*cf.* plus tard…) l'accès au contexte s'obtient via `getApplicationContext()`.
Voir aussi http://developer.android.com/reference/android/content/Context pour plus d'infos sur le contexte.

6.2. Appelez maintenant `startActivity(myIntent)` et testez.

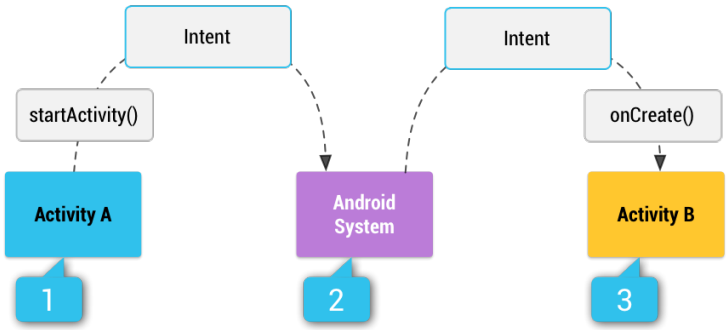
A ce stade, vous devriez voir votre deuxième activité se lancer et vous dire que vous avez réussi.

Vous venez donc d'utiliser un Intent pour demander explicitement le lancement d'une activité B à partir d'une activité A. Mais le concept d'***Intent*** est bien plus général et plus puissant que ça.

2.2 Plus généralement

Vous avez vu via la doc qu'un **Intent** est défini comme étant *« basically a passive data structure holding an abstract description of an action to be performed »*. Dit autrement, on peut voir un *Intent* comme un message asynchrone requérant une action de la part d'un autre composant applicatif[4], que ce dernier appartienne à la même application… ou pas ! C'est ce dernier scénario qui donne particulièrement son intérêt au concept d'Intent.

Dans le cas où le composant ciblé est dans la même application, l'Intent est dit **explicite** (« je souhaite lancer telle activité », c'est ce que vous venez de faire).
Dans le cas contraire, l'Intent est dit **implicite** et le choix du composant à lancer va être fait à partir d'une description de la tâche à réaliser. Si aucun composant n'est trouvé pour réaliser cette tâche, une exception est levée. Si un unique composant est trouvé, il est lancé. Si plusieurs composants sont compatibles, le choix de l'appli à lancer est proposé à l'utilisateur.



La tâche à réaliser est décrite avant tout par :

- Une **action** : une `String` décrivant de façon générique l'action à réaliser (par exemple : `ACTION_VIEW`[5] pour afficher quelque-chose à l'utilisateur – une adresse sur une carte, une photo, une page web, … – ou encore ACTION_EDIT pour éditer une donnée),
- Une **donnée**, optionnelle**,** sous la forme d'une URI, sur laquelle porte l'action.

[3] http://developer.android.com/reference/android/content/Intent#pubctors
[4] les activités en étant une des quatre catégories, nous y reviendrons.
[5] Constante de la classe `Intent`.

Q7. Tester l'utilisation d'un `Intent` pour lancer depuis votre application un navigateur afin d'afficher la page http://www.polytech.univ-tours.fr

Il est probable que la méthode `Uri.parse(String url)` vous soit utile…

En plus de l'*action* et de la *donnée*, il est parfois (souvent ?) nécessaire de préciser de façon plus détaillée la tâche à réaliser en spécifiant une catégorie et/ou des informations complémentaires sous forme d'*Extras*.

Par exemple :

```
myIntent = new Intent(Intent.ACTION_MAIN);
myIntent.addCategory(Intent.CATEGORY_APP_EMAIL);
```

produit un *Intent* pour lancer une application de gestion d'emails, et

```
myIntent = new Intent(Intent.ACTION_VIEW,
      Uri.parse("content://contacts/people/"));
```

un pour consulter la liste des contacts.

Les Intent très courants sont regroupés en catégories (*Cf.* http://developer.android.com/guide/components/intents-common) et les informations requises pour chacun d'eux sont variables.
Il est également possible, naturellement, de définir vos propres actions / catégories d'Intent…

Il arrive qu'Android ne soit pas en mesure de trouver un composant capable de prendre en charge l'action spécifiée dans l'*Intent*, que ce soit parce que l'action est mal/pas suffisamment spécifiée, soit parce qu'il n'existe aucune application capable de la réaliser. Dans ce cas, `startActivity(…)` lève une `ActivityNotFoundException`… **et votre app crashe** !
Aussi, il est (toujours !) préférable d'englober le lancement de l'activité "startActivity" d'un bloc try/catch :

```
try{
   startActivity(intent) ;
}catch(ActivityNotFoundException e){
   /*action de substitution*/
}
```

Q8. A partir de ces `Intent` courants, créez et testez successivement les Intent adéquates afin de lancer les activités pour :

8.1. Ajouter le contact ci-contre (avec validation par l'utilisateur),

John SMITH
Tel : 0123456789
Mel : john@smith.test

8.2. Envoyer par sms au « 12345 » le message « sms sent! » (avec validation par l'utilisateur),

8.3. Appeler le « 0123456 » en utilisant l'action `Intent.ACTION_DIAL`.

3 Gestion des permissions (*aka* autorisations)

Demander à une autre application de réaliser une tâche définie par un Intent spécifique ne nécessite pas de permission particulière (*cf.* par exemple Q8.1 à Q8.3). En revanche réaliser certaines actions *en interne dans votre application* ne peut se faire que lorsqu'elles ont été autorisées par l'utilisateur. C'est le cas par exemple de l'action `ACTION_CALL` qui lance un appel sans que l'utilisateur n'ait besoin d'appuyer explicitement sur « Call ».

3.1 Quatre étapes

Le processus de gestion des permissions se décompose, en version courte[6], en 4 étapes : (1) déclarer la permission, (2) vérifier que la permission est allouée, et si besoin (3) requérir la permission et (4) gérer la réponse de la requête. Voyons cela en détail.

3.1.1 Déclarer

Il convient donc pour commencer de **déclarer les permissions** dont l'application aura besoin. Cela se fait dans le fichier *AndroidManifest.xml* via autant de balises `<uses-permission>` que nécessaire, à positionner à la racine de l'élément `<manifest>`.
Par exemple

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

est requis pour toute application nécessitant un accès à Internet.

La suite du processus de gestion dépend du caractère "sensible" de la permission demandée. Android définit, outre les permissions à l'installation, principalement **2 catégories de permissions** : les permissions normales et les permissions dangereuses (*aka* permissions au runtime). Si une permission normale est déclarée, elle est allouée automatiquement par le système. C'est le cas par exemple de la permission `INTERNET` ou encore de `SET_ALARM`. Si en revanche une permission dangereuse est déclarée, il faut passer par les 3 étapes suivantes.

(*Cf.* aussi https://developer.android.com/guide/topics/permissions/overview#types pour des détails)

3.1.2 Vérifier au runtime

Depuis Android 6.0 (API 23), l'utilisateur a la possibilité à tout moment de révoquer une permission donnée antérieurement. Ainsi, en tant que développeur vous ne devez jamais préjuger de disposer d'une permission donnée, même si celle-ci a été accordée par le passé. Aussi faut-il systématiquement vérifier si vous disposez d'une permission, en appelant **`checkSelfPermission()`**, avant d'en faire usage. Systématiquement.

```
if(ContextCompat.checkSelfPermission(thisActivity,
Manifest.permission.WRITE_CALENDAR)
        == PackageManager.PERMISSION_GRANTED) {
    // Permission accordée !
}else{
    // Il faut requérir la permission, cf. étape suivante
}
```

[6] *Cf.* https://developer.android.com/guide/topics/permissions/overview pour la version longue

3.1.3 Requérir la permission et gérer la réponse de l'utilisateur

Requérir une permission se fait en deux phases : (1) en enregistrant un callback qui prendra en charge la réponse de l'utilisateur à la demande de permission, appelons-le par exemple `requestPermissionLauncher` et définissons-le comme un attribut de notre activité :

```
private ActivityResultLauncher<String> requestPermissionLauncher =
    registerForActivityResult(new RequestPermission(), isGranted -> {
        if (isGranted) {
            // Permission is granted. Continue the action or workflow in your
            // app.
        } else {
            // Explain to the user that the feature is unavailable because the
            // feature requires a permission that the user has denied. At the
            // same time, respect the user's decision. Don't link to system
            // settings in an effort to convince the user to change their
            // decision.
        }
    });
```

Dans le "if" vous placerez les actions métier à effectuer quand la permission est accordée : c'est la phase de *gestion de la réponse utilisateur*.
(2) La deuxième phase consiste à faire la demande de permission à proprement parler : appeler `launch(...)` sur l'attribut définit à la phase (1) et lui passer la permission voulue :

```
requestPermissionLauncher.launch(<REQUESTED_PERMISSION>);
```

Cet appel est donc à faire dans le `else` de l'étape "vérifier au runtime", *cf.* § précédent. On vient de mettre la dernière brique, tout s'emboîte.

Le cas où **plusieurs permissions** sont demandée simultanément est légèrement différent mais l'idée est très similaire.

4 Exercices

Nous avons vu à la question 8.3 que l'*Intent* `ACTION_DIAL` ne nécessite aucune permission. Ce n'est pas le cas de `ACTION_CALL`…

8.4. Appeler le « 0123456 » en utilisant l'action `Intent.ACTION_CALL`.

Q9. Créez et testez successivement les *Intent* adéquates afin de lancer les activités pour :

9.1. Lancer un timer qui sonnera dans 5 secondes.
9.2. *Idem* sans afficher l'IHM de l'appli de gestion du timer.

5 Bonus

Les questions/exercices ci-dessous sont optionnels en séance. A faire si vous avez pris de l'avance, ou chez vous pour réviser/aller plus loin.

Q10. Créez et testez successivement les *Intent* adéquates afin de lancer les activités pour réaliser les actions suivantes. Pensez à gérer correctement les permissions.

10.1. Allumer la lampe (flashlight) de la caméra principale.

10.2. Ouvrir une carte affichant la position de Polytech avec le label « Polytech c'est ici ! ».

Conclusion

A l'issue de cette deuxième séance, vous êtes capables de :

- Gérer les actions de « touch » de l'utilisateur et récupérer ses saisies pour agir en conséquence,
- Construire un *Intent* pour réaliser une action spécifique, en particulier une action prise en charge par un composant externe à votre application,
- Initier le passage d'un appel, l'envoi d'un SMS ou l'affichage d'une page web,
- Gérer les **permissions** pour les actions faites par vos applications Android (déclaration, vérification, requête etc.)
- Gérer la sauvegarde/restitution de couples clé-valeur via les préférences partagées,
- Utiliser un menu,
- Manipuler des versions simples de `Spinner` et `Adapter`,

6 Annexes

6.1 Références

- La plupart des images de ce document proviennent de http://developer.android.com/
- **Référence de l'API standard** : http://developer.android.com/reference/packages
- Tous les guides de la documentation officielle, par thématique : http://developer.android.com/guide

Et aussi, éventuellement :

- http://developer.android.com/courses

Plus spécifiquement sur les **permissions**

- Pour commencer : https://developer.android.com/training/permissions/requesting
- Pour aller plus loin : https://developer.android.com/guide/topics/permissions/overview