SYM Labo1 - Réponses aux questions

Vincent Guidoux, Guillaume Hochet et Marc Labie

Question 1 - Comment organiser les textes pour obtenir une application multi-langues (français, allemand, italien, langue par défaut : anglais) ? Que se passe-t-il si une traduction est manquante dans la langue par défaut ou dans une langue supplémentaire ?

On place les chaines de caractères affichées dans l'application dans un fichier `res/values/strings.xml`, celui-ci fera office de source de vérité. Si on veut ensuite ajouter une traduction, il faut créer un nouveau fichier dans le dossier `res/values-[locale]/strings.xml` en remplaçant [locale] par une locale comme fr, de... Si une string manque dans un de ces fichiers, android ira la chercher dans le fichier de base et si elle n'y existe pas, il tuera purement et simplement l'application.

Question 2 - Dans l'exemple fourni, sur le dialogue pop-up, nous affichons l'icône android. R.drawable.ic_dialog_alert , disponible dans le SDK Android mais qui n'est pas très bien adapté visuellement à notre utilisation. Nous souhaitons la remplacer avec notre propre icône, veuillez indiquer comment procéder. Dans quel(s) dossier(s) devons-nous ajouter cette image ? Décrivez brièvement la logique derrière la gestion des ressources de type « image » sur Android.

On doit tout d'abord placer l'image que l'on veut afficher dans le dossier `res/drawable/` puis on peut le référencer dans le code à l'aide de `R.drawable.mon_image`. Dans android les images sont considérées comme des ressources « **drawable** » c'est-à-dire affichable à l'écran à l'aide d'APIs. En effet, on peut par exemple prévoir plusieurs drawables pour un bouton en fonction de l'état, normal, pressé etc.

Question 3 - Lorsque le login est réussi, vous êtes censé chaîner une autre Activity en utilisant un Intent. Si je presse le bouton "Back" de l'interface Android, que puis-je constater ? Comment faire pour que l'application se comporte de manière plus logique ? Veuillez discuter de la logique derrière les activités Android.

On peut constater que l'application se ferme. Dans Android, il y a une **stack d'activité** que le système remonte lorsque l'on appuie sur le bouton back par défaut. Cette stack se remplit au fur et à mesure que de nouvelles activités sont créées dans l'application. Dans le code, quand on passait du Login à l'autre fenêtre, on appelait `finish()` sur l'activité Login ce qui la retirait de la stack. N'y restait alors que l'activité d'arrivée, donc quand on clique sur back, il n'y a rien à remonter donc l'application se fermait. Pour corriger ça, nous avons opté pour redéfinir l'action de back, en affichant un toast disant qu'on se déconnecte et en renvoyant l'utilisateur vers le Login à l'aide d'un intent. Cette fois-ci, si l'utilisateur reclique sur back, l'app se ferme car on avait bien au préalable appelé `finish()` sur l'activité précédente.

Question 4 - On pourrait imaginer une situation où cette seconde Activity fournit un résultat (par exemple l'IMEI ou une autre chaîne de caractères) que nous voudrions récupérer dans l'Activity de départ. Comment procéder ?

Il suffit de repasser des paramètres dans l'**intent** qui renvoie l'utilisateur vers le login avec `intent.putExtra()` puis la récupérer à l'aide de `intent.getStringExtra()` en testant bien que l'intent ne soit pas null.

Question 5 - Vous noterez que la méthode getDeviceId() du TelephonyManager, permettant d'obtenir l'IMEI du téléphone, est dépréciée depuis la version 26 de l'API. Veuillez discuter de ce que cela implique lors du développement et de présenter une façon d'en tenir compte avec un exemple de code.

Il faut du coup vérifier dans le code le niveau d'API du téléphone et réaliser les appels de méthode en fonction. Par exemple, dans notre application, nous réalisons notre appel dans un `if` :

```
if(Build.VERSION.SDK_INT >= 26)
    this.imei.setText(manager.getImei());
else
    this.imei.setText(manager.getDeviceId());
```

Question 6 - Dans l'activité de login, en plaçant le téléphone (ou l'émulateur) en mode paysage (landscape), nous constatons que les 2 champs de saisie ainsi que le bouton s'étendent sur toute la largeur de l'écran. Veuillez réaliser un layout spécifique au mode paysage qui permet un affichage mieux adapté et indiquer comment faire pour qu'il soit utilisé automatiquement à l'exécution.

Voir le nouveau fichier de layout dans `/res/layout-land/authent.xml`. Chaque activité se base sur un nom de fichier de layout, comme `authent.xml`. Pour créer des layouts spécifiques à l'orientation du téléphone, on crée un nouveau fichier authent.xml (pour l'activité d'authentification) qu'on place cette fois dans un dossier `/res/layout-land`, pour landscape.

Question 7 - Le layout de l'interface utilisateur de l'activité de login qui vous a été fourni a été réalisé avec un LinearLayout à la racine. Nous vous demandons de réaliser un layout équivalent utilisant cette fois-ci un RelativeLayout.

Voir le fichier de layout dans `/res/layout/authent.xml`. Le **RelativeLayout** place les éléments relativement aux autres. Nous plaçons donc le premier input en haut de l'écran, le second en-dessous du premier et le bouton en-dessous du deuxième input.

Question 8 - Implémenter dans votre code les méthodes onCreate(), onStart(), onResume(), onPause(), onStop(), etc… qui marquent le cycle de vie d'une application Android, et tracez leur exécution dans le logcat. Décrivez brièvement à quelles occasions ces méthodes sont invoquées. Si vous aviez (par exemple) une connexion Bluetooth (ou des connexions bases de données, ou des capteurs activés) ouverte dans votre Activity, que faudrait-il peut-être faire, à votre avis (nous ne vous demandons pas de code ici) ?

Les méthodes de cycle de vie sont :

- **onCreate** : Appelée lorsque android crée l'activité, on y réalise la logique qui ne doit être exécutée qu'une fois dans la vie de l'activité.
- **onStart** : lorsque l'activité devient visible à l'utilisateur, on peut y appeler le code qui gère la maintenance de l'UI.
- **onResume** : lorsque l'utilisateur peut intéragir avec l'activité. Elle reste dans cet état tant qu'elle ne perd pas le focus de l'utilisateur
- **onPause** : appelée par android au premier signe que l'utilisateur s'apprête à quitter l'activité, par exemple si une modal apparaît. L'activité peut toujours être partiellement visible mais n'a plus le focus
- **onStop** : lorsque l'activité n'est plus visible par l'utilisateur. Utile pour libérer des ressources non nécessaires lorsque l'activité n'est pas visible
- **onDestroy** : appelée juste avant que l'activité soit détruite, à cause d'un appel à finish(), ou lorsque le natel change d'orientation par exemple. On devrait y libérer toutes les ressources qui ne l'ont pas encore été.