Labo 2 DAA

Auteurs : Junod Arthur, Dunant Guillaume, Häffner Edwin

Les activités

Implémentation

L'activité de base se trouve dans le fichier `MainActivity`. Lorsqu'on clique sur le bouton "Editer", nous utilisons la librairie AndroidX pour créer une nouvelle activité `UsernameInputActivity` qui va inviter l'utilisateur à entrer ou bien modifier son nom. L'utilisation de la librairie se fait dans la classe `PickNameContract` qui va instancier une `Intent` et qui va attendre le résultat de celle-ci pour la renvoyer à l'activité initiale.

Questions

> Que se passe-t-il si l'utilisateur appuie sur « back » lorsqu'il se trouve sur la seconde Activité ?

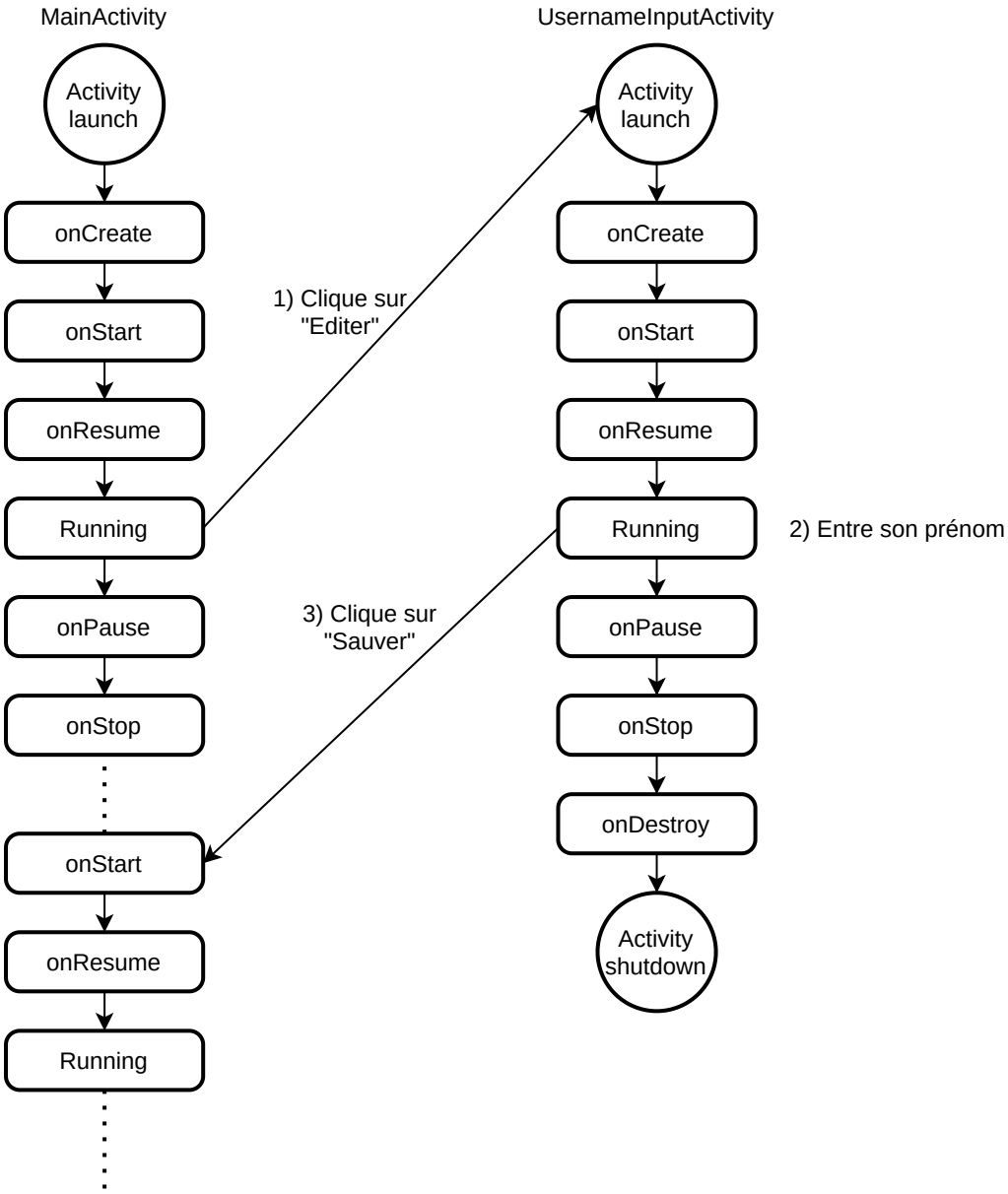
On obtient le texte "Bienvenue null !". Nous obtenons cet affichage car dans le contrat que nous avions défini au début, nous retournions `null` si l'activité ne se finissait pas "correctement". Et elle ne se finit pas correctement si nous n'appuyons pas sur le bouton "sauver".

Pour résoudre ce problème on peut ou bien modifier la valeur de retour dans le contrat ou, comme nous avons fait, faire une vérification du retour du contrat afin de vérifier qu'il ne soit pas `null` afin de choisir l'affichage en conséquence (ici, ne rien changer à la `String` de base ou changer l'affichage selon le nom sauvegardé).

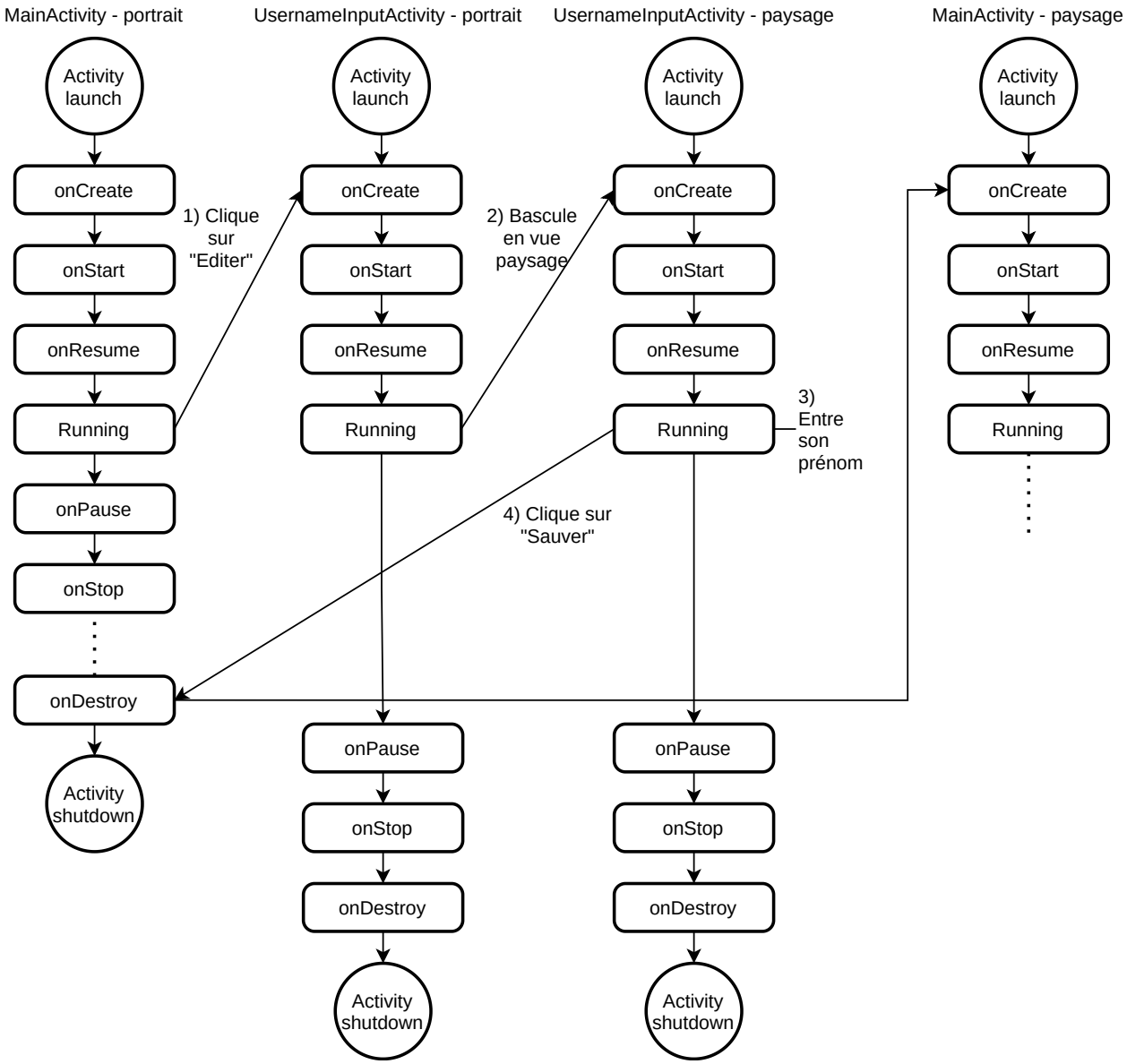
> Veuillez réaliser un diagramme des changements d’état des deux Activités pour les utilisations suivantes, vous mettrez en évidence les différentes instances de chaque Activité

> - L’utilisateur ouvre l’application, clique sur le bouton éditer, renseigne son prénom et sauve.

Diagramme d'activité :



> - L’utilisateur ouvre l’application en mode portrait, clique sur le bouton éditer, bascule en mode paysage, renseigne son prénom et sauve.



> Que faut-il mettre en place pour que vos Activités supportent la rotation de l’écran ? Est-ce nécessaire de le réaliser pour les deux Activités, quelle est la différence ?

Il faut sauvegarder l'état de notre message de bienvenue dans un `Bundle` à la destruction de la première activité `Main` (`onSaveInstanceState()`) puis, si un `Bundle` existe, récupérer son état à la création d'une nouvelle activité (`onCreate()`). Car quand nous pivotons notre écran, une nouvelle activité est créés pour l'affichage en paysage qui va exécuter le même code.

La deuxième activité, où l'on peut éditer notre nom, ne comporte pas le même problème car elle contient non pas un `TextField` mais un `EditText` qui est gardé en mémoire dans la `View` ce qui évite de perdre sa valeur quand nous pivotons et créons une nouvelle activité.

Les fragments

Implémentation

L'implémentation de l'exercice sur les fragments se trouve dans la *main_activity2*. Pour afficher les deux fragments, nous avons du adapter le nom du package des fragments à notre projet puis les importer en tant que widget grâce au *FragmentContainerView*.

Questions

> Les deux Fragments fournis implémentent la restauration de leur état. Si on enlève la sauvegarde de l'état sur le ColorFragment sa couleur sera tout de même restaurée, comment pouvons-nous expliquer cela ?

La couleur sera quand même restaurée car la modification des sliders va changer la couleur de la vue racine, et donc la couleur est également stockée dans cette vue est pas uniquement dans le fragment.

> Si nous plaçons deux fois le CounterFragment dans l'Activité, nous aurons deux instances indépendantes de celui-ci. Comment est-ce que la restauration de l'état se passe en cas de rotation de l'écran ?

Le fragment *CounterFragment* surcharge la méthode *onSaveInstanceState* ce qui permet à celui-ci de sauvegarder son état (dans notre cas la valeur du compteur) dans un "Bundle". Les valeurs dans le bundle seront disponibles dans les méthodes qui créent le fragment (*onCreate* et *onViewCreated*).

Lors d'une rotation de l'écran, le fragment est stoppé puis redémarré, ce qui invoque les méthodes citées précédement.

Le fragment manager

Implémentation

l'ActivitiyFragment

Nous avons une nouvelle classe Kotlin `ActivityFragment` avec son layout dans `activity_fragment.xml` qui nous permet de gérer la "backStack" remplie (ou non) des différents `fragmentStep`.

Quand nous appuyons sur le bouton "next", nous appelons la function `replaceFragment()` sur le fragment manager qui lance une transaction `replace().addToBackStack()` qui va remplacer le fragment dans la `FragmentContainerView` et le lui rajouter à sa "backStack".

Le bouton "back" va pop la "backStack" du fragment manager afin de retrouver le fragment précédent.

Le bouton "close" fait juste appel à `finish()` afin de fermer l'application.

Le fragment step

Le fragment `step` récupère le compte de la "backStack" de son élément parent en passant par le `parentFragmentManager`, cela permet de facilement remplacer le numéro affiché dans ce fragment par le compte récupéré.

Question

> A l'initialisation de l'Activité, comment peut-on faire en sorte que la première étape s'affiche automatiquement ?

On fait un appel a une transaction sur le fragment manager dans le `onCreate()` afin de `replace()` le fragment par la première étape s'il n'y pas déjà de fragment à afficher.

> Comment pouvez-vous faire en sorte que votre implémentation supporte la rotation de l'écran ? Nous nous intéressons en particulier au maintien de l'état de la pile de Fragments et de l'étape en cours lors de la rotation.

Nous passons le contexte au layout de l'activité hôte. Le fragment manager sera gardé entre les rotations car celui-ci n'est pas détruit et `FragmentContainerView` pourra, du coup, récupérer le fragment à afficher dans le fragment manager quand nous appelons `OnViewCreated`.

> Dans une transaction sur le Fragment, quelle est la différence entre les méthodes add et replace ?

La transaction `replace()` fait 2 choses, elle `remove()` tous les fragments actuels de la `ContainerView` et `add()` le nouveau fragment donné à celle-ci.

Si l'on appel que `add()` les fragment vont se superposer à l'affichage car ils ne seront pas enlevés (mais ils le seront tout de même, un à un, si l'on "pop" la "backStack").