DAA - Laboratoire 2

April 24, 2024

Émilie Bressoud **Sacha Butty** **Loic Herman**

1 Questions

1.1 Les Activités

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

Lorsque l'utilisateur appuie sur « back » dans la seconde Activité (InputNameActivity), l'activité se termine sans renvoyer de résultat à la première Activité (`WelcomeActivity`). La `WelcomeActivity` reprend le premier plan sans modification du nom d'utilisateur.

- *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 :*

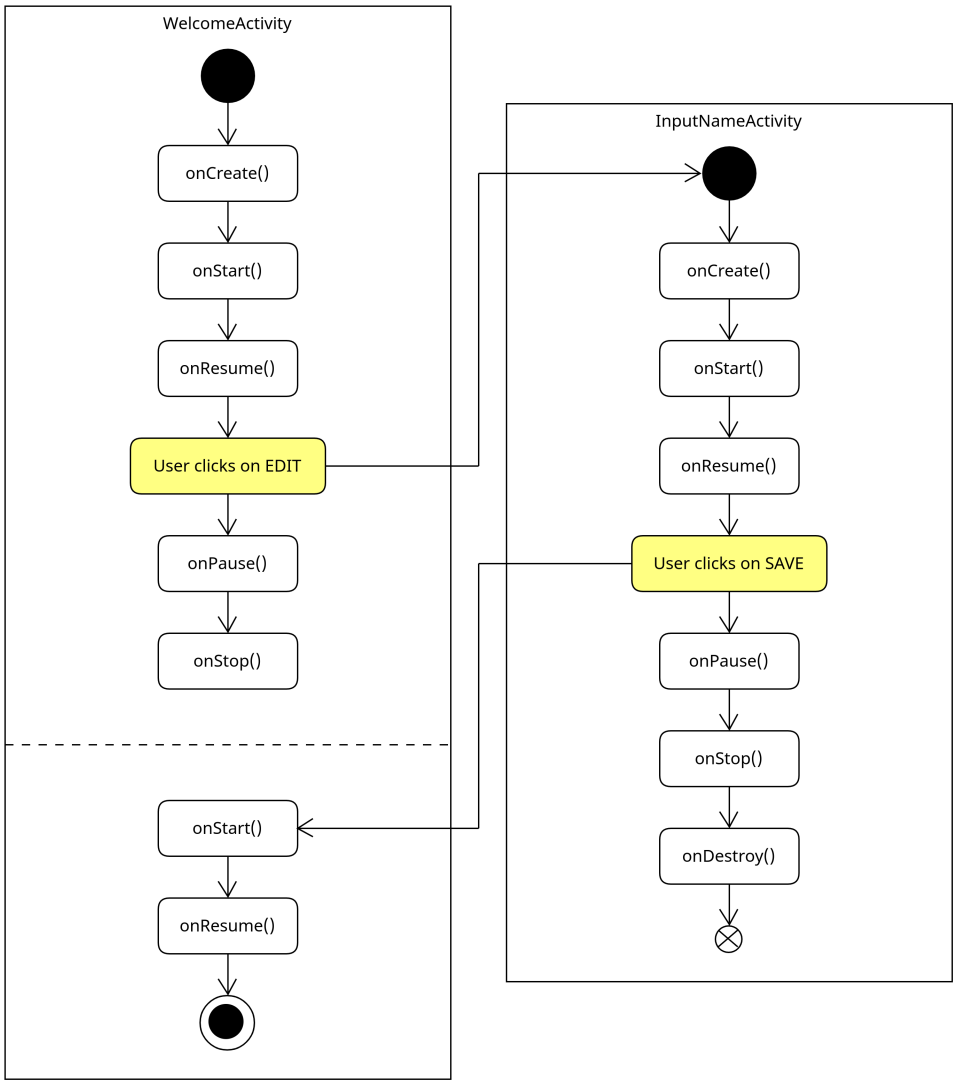

Fig. 1. – L'utilisateur ouvre l'application, clique sur le bouton éditer, renseigne son prénom et sauve

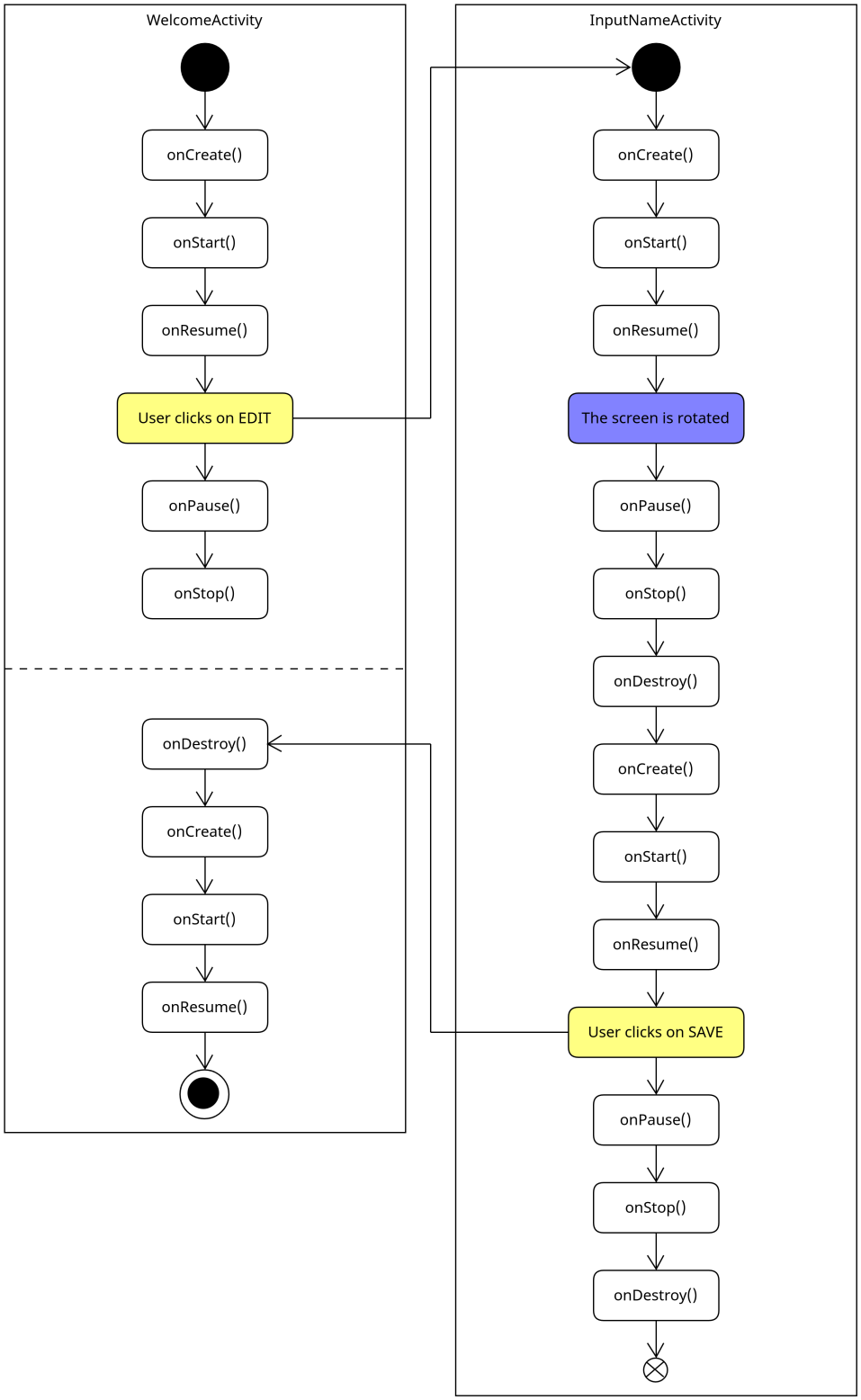

Fig. 2. – L'utilisateur ouvre l'application en mode portrait, clique sur le bouton éditer, bascule en mode paysage, renseigne son prénom et sauve.

- *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 ?*

Pour supporter la rotation de l'écran, il faut mettre en place :

- La sauvegarde de l'état de l'activité dans `onSaveInstanceState()`
- La restauration de l'état dans `onCreate()` ou `onRestoreInstanceState()`

Il n'est cependant pas nécessaire d'implémenter la sauvegarde et la restauration pour l'activité `InputNameActivity`, car le contenu du champ `TextView` est automatiquement sauvegardé dans son instance et pas dans l'activité qui l'héberge.

Pour l'activité `WelcomeActivity` il est du coup nécessaire de sauvegarder l'état et de le restaurer correctement, puisque la valeur du champ est sauvegardée dans un champ de la classe qui sera recréée lors d'une rotation de l'écran.

1.2 Les Fragments

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 ?

Android sauvegarde l'état automatiquement pour certains composants et vues. Les `SeekBar` qui sont utilisés sur lesquels un event listener est enregistré pour le changement de la couleur du fond en font partie.

Donc, lors d'un changement de la configuration (par exemple via une rotation de l'écran), en plaçant un break-point sur l'event listener fourni au `SeekBar` nous pouvons bien constater que l'événement est relancé lorsque l'activité est recréée.

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 ?

Chaque `FragmentContainerView` aura une instance dédiée du fragment défini, donc Android fera la distinction entre les deux lors de la sauvegarde de l'état. Même si le contenu de la vue est partagé, les instances sont bien séparées et ne partagent aucun état commun.

1.3 Fragments Manager

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

Pour afficher la première étape, il suffit de créer le tout premier fragment dans la fonction `onCreate` de l'activité. Cela est fait en appelant la fonction `nextStep` qui crée le premier fragment.

1.3.2 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.

Pour supporter la rotation de l'écran, nous utilisons le `FragmentManager` pour gérer la pile de Fragments. Lorsque l'écran est rotatif, le système appellera la méthode `onSaveInstanceState` de l'Activité, qui sauvegardera l'état de la pile de Fragments.

Afin de ne pas créer un fragment qui incrémente le step lors de la rotation, il suffit de vérifier dans la méthode `onCreate` contenant `nextStep` qu'il n'y a pas déjà une instance sauvée. Dans le cas échéant, on ignore la création de l'incrémentation du step. Cela recrée simplement un fragment avec le step courant qui a été sauvegardée avec l'argument `ARG_STEP`

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

La méthode `add` ajoute un nouveau fragment et garde les fragments existants, tandis que la méthode `replace` enlève le fragment existant et ajoute le nouveau fragment.

2 Description de l'implémentation

2.1 Activités

Pour cette première manipulation, nous avons repris le layout fourni par le template en rajoutant ce qui était demandé. Pour la communication entre les deux activités, nous utilisons un `Contract` qui offre une approche typée pour l'attente du résultat.

Nous avons défini selon ce qui était demandé une classe abstraite `LoggableCompatActivity` qui override les fonctions de lifecycle demandées pour rajouter des lignes de logs.

2.1.1 Vue d'ensemble

L'implémentation se compose de plusieurs fichiers Kotlin qui fonctionnent ensemble pour créer une application Android simple avec un écran d'accueil et une fonctionnalité de saisie de nom d'utilisateur. Les composants principaux sont :

- `WelcomeActivity` : L'écran principal de l'application
- `InputNameActivity` : Gère la saisie du nom d'utilisateur
- `UsernameInputContract` : Gère la communication entre les activités
- `LoggableCompatActivity` : Une classe de support pour l'enregistrement des événements du cycle de vie des activités

2.1.2 WelcomeActivity

La `WelcomeActivity` est le point d'entrée pour cette manipulation. Elle hérite de `LoggableCompatActivity` et implémente les fonctionnalités suivantes :

- Affiche un message de bienvenue avec ou sans nom d'utilisateur
- Fournit un bouton « Modifier » pour modifier le nom d'utilisateur
- Gère la préservation et la restauration de l'état

Détails d'implémentation clés :
- Utilise `ActivityResultContract` pour lancer `InputNameActivity` et recevoir le résultat
- Implémente `onSaveInstanceState` et `onRestoreInstanceState` pour gérer les changements de configuration
- Met à jour le message de bienvenue en fonction de la présence ou de l'absence d'un nom d'utilisateur

2.1.3 InputNameActivity

L'`InputNameActivity` est responsable de la saisie du nom d'utilisateur. Elle hérite également de `LoggableCompatActivity` et inclut :

- Un `EditText` pour la saisie du nom d'utilisateur
- Un bouton « Enregistrer » pour confirmer la saisie
- Une logique pour renvoyer le nom d'utilisateur saisi à `WelcomeActivity`

Détails d'implémentation clés :
- Utilise `Intent` pour recevoir et renvoyer des données à `WelcomeActivity`
- Implémente une fonction `closeActivity` pour gérer la configuration du résultat et la fermeture de l'activité

2.1.4 UsernameInputContract

La classe `UsernameInputContract` étend `ActivityResultContract` pour gérer la communication entre `WelcomeActivity` et `InputNameActivity`. Elle définit :

- Type d'entrée : `String?` (String nullable)
- Type de sortie : `String?` (String nullable)
- `createIntent` : Crée l'intent pour lancer `InputNameActivity`
- `parseResult` : Gère le résultat renvoyé par `InputNameActivity`

2.1.5 LoggableCompatActivity

Cette classe de support étend `AppCompatActivity` et surcharge les méthodes du cycle de vie pour enregistrer les changements d'état. Nous avons choisi de définir cette classe pour permettre d'abstraire la gestion des logs pour les classes de l'activités à un endroit. Elle inclut :

- Méthodes surchargées : `onCreate`, `onStart`, `onResume`, `onPause`, `onStop`, `onDestroy`
- Enregistrement utilisant `Log.d` d'Android avec un tag « [STATE CHANGE] »

2.2 Fragments

L'implémentation pour la partie 2 de l'exercice consiste en une seule activité, `MainActivityFragment1`, qui est conçue pour héberger deux fragments. Cette approche permet d'afficher deux composants d'interface utilisateur indépendants au sein d'une même activité.

La `MainActivityFragment1` est l'activité principale pour cette partie de l'exercice. Elle hérite de `AppCompatActivity` et initialise le layout de l'activité via la fonction surchargée `onCreate`.

En terme de layout, l'activité fait simplement appel à deux `FragmentContainerView` qui s'occupent de faire le rendu des fragments fournis.

2.3 Fragments Manager

Pour le fragment manager, une classe `MainActivityFragment2` a été créée et a été ajoutée au `MainActivity`. Elle hérite d'`AppCompatActivity` et utilise :

- `onCreate`: hérite de la classe parent et set le layout `activity_main_fragment2` contenant les boutons de navigation.
- Des fonctions permettant la navigation entre les étapes et sont appelées lors d'un clic sur le bouton correspondant
  - `back`: supprime le dernier fragment de la pile grâce à `supportFragmentManager.popBackStack()` et affiche l'étape précédente.
  - `close` : ferme l'activité `activity_main_fragment2`.
  - `next` : ajoute un nouveau fragment à la pile et affiche l'étape suivante.
- `nextStep`: une fonction qui crée un nouveau fragment à chaque appel, utilisé dans `onCreate` afin de créer le tout premier fragment (step 0) si l'activité a été créer pour la première fois seulement et dans `next` pour la création de nouveaux fragments. Utilisation de `backStackEntryCount` afin de connaître le nombre de fragment créé.

Le layout de cette activité contient aussi le `FragmentContainerView` qui mappe le layout du fragment `fragment_step`.

Le layout `fragment_step` contient simplement un `TextView` affichant l'étape et dont le contexte provient du fragment `StepFragment`

La classe `StepFragment` hérite de Fragment et contient:
- `newInstance`: incrémentation du step à chaque fois qu'un nouveau fragment est créé.
- `onCreate`: set la nouvelle valeur dans step
- `onCreateView`: crée la view à partir du layout fragment_step
- `onViewCreated`: change le texte à afficher à partir du step courant.
- `onSaveInstanceState`: Sauvegarde l'état du fragment
- `onViewStateRestored`: Récupération du step si un fragment a été sauvegardé