1 Création du programme de base

Créez un nouveau projet de nom `DemoCycle` selon les consignes suivantes :

1. Il n'y aura qu'une activité dans l'application. Celle ci comportera :
   - Une zone de saisie numérique d'id `increment`.
   - deux zones d'affichage (nommées `cumul` et `inverse`) (affichage par défaut '1' et '1').
   - Un bouton 'Go'
2. L'activité sera également doté d'un attribut entier que nous appellerons *valeur* et initialisé à `1`.
3. Le comportement de l'application sera le suivant : à chaque appui sur le bouton, le programme récupère la valeur saisie et l'additionne à la valeur précédente de *valeur*. Puis celle-ci est affichée dans la zone `cumul` tandis que sont inverse sera affiché dans la zone `inverse`. Enfin, la zone de saisie est remise à zéro.

Exemple : après les saisies de 2, puis de 5, l'application aura l'apparence suivante (peu importe l'orientation).

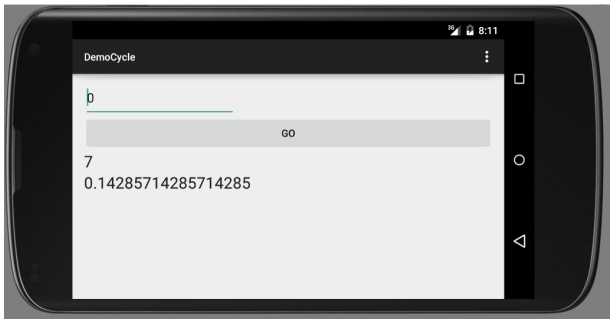

FIGURE 1 – Apparence de l'application DemoCycle

1.1 Écrivez et testez ce projet

2 En cas de plantage...

Vous avez peut être, lors de vos tests, tenté d'appuyer sur le bouton `Ajouter` alors que la zone de saisie était vide. Si ce n'est pas la cas, faîtes le. Que se passe t il alors ?

2.1 Localiser l'erreur

Vous pouvez localiser une fenêtre de nom *`logcat`* dans votre environnement. Celle-ci affiche la liste des exception imbriquées qui se sont produites. Retrouvez la ligne qui correspond à votre propre code et déduisez en la ligne fautive.

2.2 Outils de mise au point

Placez un point d'arrêt sur la ligne repérée par un clic sur la colonne à gauche de la ligne. Ceci fait apparaître un cercle rouge (cf. figure 2).

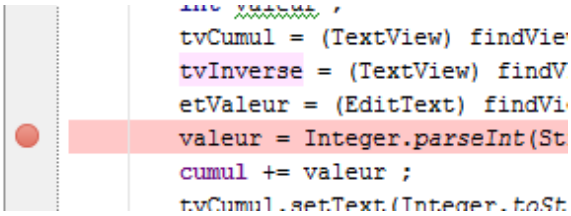

FIGURE 2 – Établissement d'un point d'arrêt.

Puis lancer le programme **en mod debug** (*`Run/Debug app`*) .
Sur l'émulateur, laissez la zone de saisie vide et appuyez sur le bouton. Que se passe t-il ? Notez que vous pouvez alors :
- Afficher la valeur d'une variable.
- Exécuter une instruction.
- Reprendre l'exécution du programme.
- ...
-

3 Etude du cycle de vie de l'activité

3.1 Observation

Arrêtez l'exécution du programme (*Run/Stop)* .
Nous allons profiter de ces facilités pour observer le cycle de vie des programmes. Pour cela :

1. Surchargez la méthodes événementielle `onStart` en incluant dans le corps de cette méthode un simple appel à la méthode de base :

```
public void onStart() {                    1
        super.onStart();                   2
}                                          3
```

2. Faîtes de même pour les méthode `onStop`, `onDestroy`, `onPause`, `onResume`, `onSaveInstanceState` [1], `onRestoreInstancesState` [2], `onRestart`
3. Placez un point d'arrêt dans chacune de ces méthodes ainsi que dans la méthode `onCreate`.

Le programme s'arrétera lorsqu'il rencontrera un point d'arrêt. Ceci vous permet d'observer l'appel à une méthode. Vous pourrez ensuite reprendre l'exécution du programme (*Run/Resume Program).* Vous pourrez ainsi, en lançant l'exécution tracer tous les appels que l'environnement adresse à l'activity en fonction des circonstances.
Que se passe t il dans les scénarios suivant :

a) i. Au lancement du programme.
ii. Puis, une fois le programme lancé, rappelez l'écran d'accueil (bouton central de l'interface).
iii. Puis, relancez le programme (via l'interface android).

b) i. Après lancement du programme, faîtes pivoter le terminal (virtuel ou non).

c) i. Dans les paramétrages du terminal (`Settings`, cochez "Don't keep activities" (Figure 3).
ii. Recommencez le scénarion `a`.

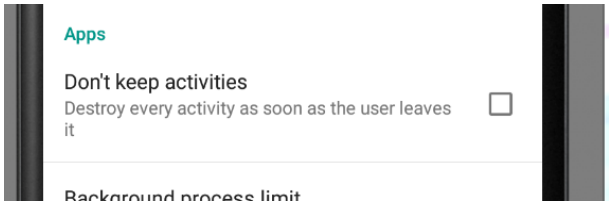

FIGURE 3 – Réglages android.

d) i. Ajoutez un bouton 'Appel à un ami' à votre activité.
ii. Ajoutez à votre code l'imports suivants :

```
import android.provider.ContactsContract.Contacts;          1
```

iii. Connectez le nouveau bouton à l'exécution du code suivant :

```
Intent contactPickerIntent = new Intent(Intent.ACTION_PICK,          1
                Contacts.CONTENT_URI);                               2
startActivityForResult(contactPickerIntent, 1001) ;                  3
```

Ce code à pour objet d'appeler l'application Contact permettant à l'utilisateur de choisir l'un de ses contact.

3.2 Sauvegarde et restauration des données

Corrigez votre programme pour que les données soient conservées lors de ces différents scénarii.. (inspirez-vous du cours).

4 Les adapters

4.1 Mise en place d'un adapter

Créez une nouvelle application. Cette dernière aura pour objet de permettre à l'enseignant de garder la trace des appareils prêtés aux étudiants lors des TP. Il s'agit de pouvoir enregistrer la trace de l'association entre un étudiant et un appareil. La listes des appareils disponible est connue de même que la liste des étudiants.
Créez dans cette application dans le fichier*strings.xml* les ressources (tableau de chaînes) suivantes :

1. attention, cette méthode a un paramètre
2. attention, cette méthode a un paramètre

```
<string-array name="tablets">
    <item>tablette 01</item>
    <item>tablette 02</item>
    <item>tablette 03</item>
    <item>tablette 04</item>
    <item>tablette 05</item>
    <item>tablette 06</item>
</string-array>
<string-array name="students">
    <item>DISPO</item>
    <item>alain</item>
    <item>bruno</item>
    <item>charles</item>
    <item>denis</item>
    <item>eric</item>
    <item>flavien</item>
</string-array>
```

Dans l'activité vierge, créez un composant de type `spinner`. et dont l'id est *terminal*.
Ajoutez les lignes suivantes au code, dans la méthode `onCreate` de l'activité, **Après l'instruction** `setContentView`.

```
Spinner spTerminal = (Spinner) findViewById(R.id.terminal);                    1
ArrayAdapter adTerminal = ArrayAdapter.createFromResource(this, R.array.tablets,  2
            android.R.layout.simple_spinner_item) ;                            3
spTerminal.setAdapter(adTerminal) ;                                            4
```

4.2 Répétez la même opération pour un spinner *student*

4.3 Ajouter deux champs qui reflètent les valeurs sélectionnées par l'utilisateur

La sélection d'un item par un spinner (tablette ou étudiant) doit changer l'affichage du champ correspondant.
La documentation en ligne "Android developer" vous sera d'une aide précieuse!!

5 Préparons le modèle.

L'application finale peut être décrite selon le modèle suivant :
– On définit une assocation comme un couple intégrant un nom d'étudiant et une tablette.
– Le modèle compet est représenté par une liste d'associations.
Définir la classe `Modele` (purement java) permettant les opérations suivantes :
– Constructeur par défaut (lecture de la ressource `tablets` et initialisation des étudiants à 0 (`DISPO`)
– `getCount()` retourne le nombre de tablettes
– `getStudent(i)` retourne l'étudiant qui a obtenu la tablette `i` (0 si tablette dispo).
– `getTerminal(i)` retourne le nom de la tablette `i`.
– `setStudent(i,e)` attribue la tablette `i` à l'étudiant `e` (retourne `true` si ok , `false` sinon (si l'étudiant a déja une autre tablette par exemple).
– `save(Bundle outState)` sauvegarde l'tétat du modèle vers le bundle.
– un constructeur `Modele(Bundle inState)` reconstruit le modele à partir du Bundle.

6 Mise en forme

– Remplacez tous les composants de l'activité par un composant `ListView`.
– Créez un nouveau layout qui détermine l'apparence d'une entrée de la liste (à savoit terminal (Large Text) et etudiant (Spinner).
– Reprenez l'adapter pour le Spinner et Ecrivez l'adapteur pour la ListView.
– Ajoutez le listener qui met à jour le modèle quand on change la sélection du Spinner (`OnItemSelectedListener`).
Vérifiez qu'aucune donnée n'est perdu lors d'un changement d'activité.
Comtemploez le résultat (fig. 4).

7 Bonus.

Utilisez un Toast pour signaler l'erreur (quand un étudiant est sélectionné deux fois).

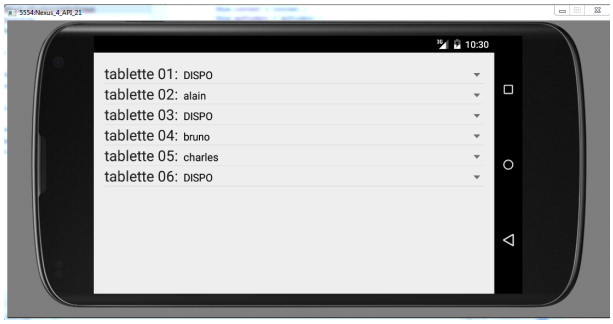

FIGURE 4 – Résultat final.