TP : cycle de vie d'une Activité

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_multi);
}

protected void onStart() {..

protected void onRestart() {..

protected void onResume() {..

protected void onPause() {..

protected void onStop() {..

protected void onDestroy() {..
```

Plan

- reprise de l'application Multiplication créée lors des séances précédentes
- nous allons observer le cycle de vie d'une activité
- pour cela, modification de votre programma java pour ajouter les méthodes de callback

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_multi);
}

protected void onStart() {

protected void onRestart() {

protected void onResume() {

protected void onPause() {

protected void onStop() {

protected void onDestroy() {
```

- comment savoir quelle méthode est appelée et quand ?

  → en utilisant des Toast

Activity - Cycle de vie

Toast

- c'est un Widget
- Création : Toast.makeText(Context, String, int)
  - Context → getApplicationContext() ou this (puisque Activity est une sous-classe de Context).
  - String correspond au message
  - la durée d'affichage de la notification est donnée par Toast.LENGTH_LONG or SHORT
- position du Toast → default ou setGravity(Gravity.POSITION,xoffset,yoffset)
- pour l'affichage : show()

Dans votre fichier java

```
protected void onResume() {
    super.onResume();
    Toast myToast;
    myToast = Toast.makeText(this, new String("onResume()"), Toast.LENGTH_LONG);
    myToast.setGravity(Gravity.CENTER, 0, 0);
    myToast.show();

}
```

- Appliquez la même chose à toutes les méthodes onXXX()

Activity - Cycle de vie

résultats de l'exécution de l'application

- lorsque vous cliquez sur l'icône l'app démarre
  → onCreate() - onStart() - onResume()
- état : *Running*
- lorsque vous cliquez sur le bouton *Home* (vous quittez temporairement l'app)
  → *onPause() - onStop()*
- état : *Stopped*
- retour à l'application
  → *onRestart() - onStart() - onResume()*
- état : *Running*
- cliquez sur le bouton "Exit"
  → *onPause() - onStop() - onDestroy()*
- fin de l'application

qu'en est-il des variables ?

- *onPause() - onStop()* $\rightarrow$ *onRestart() - onStart() - onResume()*
- nous allons utiliser une variable pour compter chaque appel à une *callback method*
- nous allons utiliser les Toast pour afficher la valeur de cette variable

```
protected void onResume() {
    super.onResume();
    nbOfCalls++;
    Toast myToast = Toast.makeText(this, new String("onResume() "+nbOfCalls), Toast.LENGT
    myToast.setGravity(Gravity.CENTER, 0, 0);
    myToast.show();
}
```

- allez-y, faites-le

qu'en est-il des variables ?

- *onPause() - onStop()* $\rightarrow$ *onRestart() - onStart() - onResume()*
- nous allons utiliser une variable pour compter chaque appel à une *callback method*
- nous allons utiliser les Toast pour afficher la valeur de cette variable

```
protected void onResume() {
    super.onResume();
    nbOfCalls++;
    Toast myToast = Toast.makeText(this, new String("onResume() "+nbOfCalls), Toast.LENGTH_LONG);
    myToast.setGravity(Gravity.CENTER, 0, 0);
    myToast.show();
}
```

- exécution !! **ATTENTION à ne pas changer l'orientation de votre téléphone durant l'exécution**
- normalement la valeur est mise à jour

Activity - Cycle de vie

résultat de l'exécution de l'application

- lorsque vous cliquez sur l'icône l'app démarre
  → *onCreate() 1 - onStart() 2 - onResume() 3*
- état : *Running*
- lorsque vous cliquez sur le bouton *Home* (vous quittez temporairement l'app)
  → *onPause() 4 - onStop() 5*
- état : *Stopped*
- retour à l'application
  → *onRestart() 6 - onStart() 7 - onResume() 8*
- état : *Running*
- cliquez sur le bouton "Exit"
  → *onPause() 9 - onStop() 10 - onDestroy() 11*
- fin de l'application

Activity - Cycle de vie

qu'en est-il des variables ?

- les variables sont sauvegardées lorsque l'Activity est stoppée
- donc, lorsque l'Activity est dans l'état *Paused* ou *Stopped*

  $\rightarrow$ ses données sont conservées dans la mémoire du système
- lorsque l'Activity est de nouveau au premier plan elle retrouve ses données intactes

cependant... exécutez de nouveau l'app mais en **changeant l'orientation** de votre téléphone plusieurs fois **pendant l'exécution**

Activity - Cycle de vie

changement d'orientation $\Rightarrow$ destruction et recréation de l'app

- changement de l'orientation du smartphone $\rightarrow$ la variable *nbOfCalls* redémarre du début...
- en effet, **l'app est détruite et recréée instantanément !**
- catastrophe !!
- pour éviter la perte des données (score en cours, valeurs temporaires, etc.), vous pouvez empêcher le changement d'orientation ou sauvegarder l'état de l'instance de votre Activity.

Activity - Cycle de vie

sauvegarde et restauration de l'état de l'instance de l'Activity

- une Activity détruite ne peut pas retrouver seule son état avant sa destruction

  $\Rightarrow$ il faut sauvegarder les données
- Android fournit des méthodes pour le faire :
  - sauvegarde avec la méthode onSaveInstanceState()
  - restauration avec la méthode onRestoreSInstanceState()

Views

- l'état des *Views* est automatiquement sauvegardé à la condition qu'elles aient un identifiant ID (android :id)

Activity - Cycle de vie

onSaveInstanceState()

- appel : onSaveInstanceState(Bundle theBundle)

  le système fournit Bundle (une sorte de "sac à données")

  il peut être utilisé pour stocker les données selon un format de type clef-valeur :

  - String : theBundle.putString("nomdelachaine",valeur) ou
  - Integer : theBundle.putInt("nomdelentier",valeur)

onRestoreInstanceState()

- appel : onRestoreInstanceState(Bundle sacDeDonnees)
  - sacDeDonnees.getString("nomdelachaine")
  - sacDeDonnees.getInt("nomdelentier")

Activity - Cycle de vie

fichier java

```
public void onSaveInstanceState(Bundle bagOfData) {
    bagOfData.putInt("nb_calls", nbOfCalls);
    super.onSaveInstanceState(bagOfData);
}

public void onRestoreInstanceState(Bundle bagOfData) {
    super.onRestoreInstanceState(bagOfData);
    nbOfCalls = bagOfData.getInt("nb_calls");
}
```

1. exécutez de nouveau l'app (n'oubliez pas de changer l'orientation)
2. sauvegardez de même les opérandes de votre multiplication pour qu'un changement d'orientation ne modifie pas l'opération

Activity - Cycle de vie - résumé

- Activity : un composant dont le but est de permettre l'interaction avec l'utilisateur
- Une Activity possède un cycle de vie qui commence avec un appel à la méthode *onCreate()* (méthode requise) et qui se termine avec la méthode *onDestroy()* (qui peut être omise)
- Une Activity affiche une GUI produite par la méthode *setContentView()* selon le principe de la désérialisation à partir de données stockées dans le répertoire res (ressources) dans des fichiers XML
- chaque changement dans la configuration de l'app $\rightarrow$ destruction $\rightarrow$ recréation

  $\Rightarrow$ les données doivent être sauvegardées en implémentant les méthodes *onSaveInstanceState()* et *onRestoreInstanceState()*

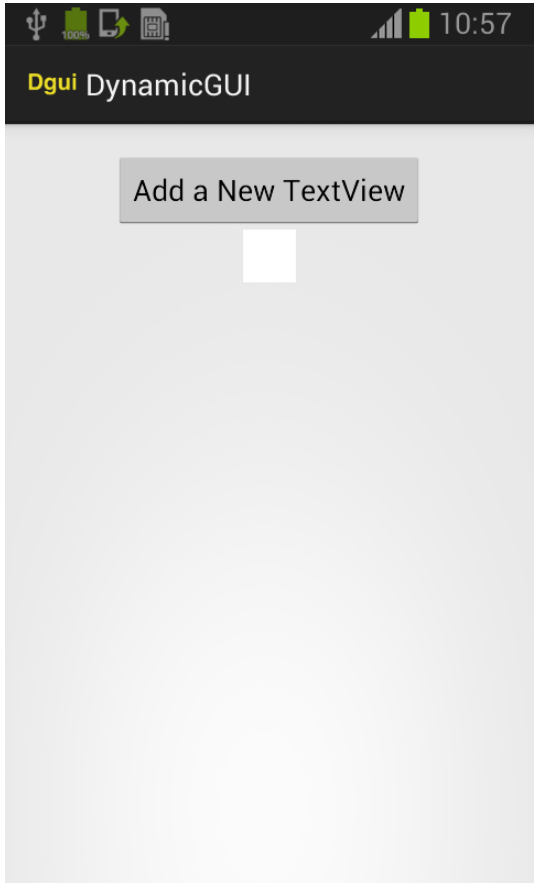
Références

- http ://developer.android.com/training/basics/activity-lifecycle/index.html
- http ://developer.android.com/reference/android/app/Activity.html
- http ://developer.android.com/training/basics/activity-lifecycle/starting.html
- http ://developer.android.com/training/basics/activity-lifecycle/pausing.html
- http ://developer.android.com/training/basics/activity-lifecycle/stopping.html
- http ://developer.android.com/training/basics/activity-lifecycle/recreating.html
- http ://developer.android.com/guide/topics/ui/notifiers/toasts.html

Exercice

GUI dynamique

- création d'un nouveau projet : DynamicGUI
- cette GUI est composée d'un LinearLayout avec orientation verticale. Ce layout contient :
  - un bouton "Add a New TextView"
  - un GridLayout (initialement vide mais formé de 3 colonnes



pour le GridLayout voir https://di.iut.univ-lehavre.fr/pedago/info2/M4104C_ProgMobile/CoursTP/cours-gui-et-exo.pdf

Action du bouton

- lorsque le bouton est pressé, un nouveau *TextView* est ajouté au GridLayout
- si le nombre de *TextView* est égal à 9, un nombre aléatoire de *TextView* est supprimé (voir l'image ci-dessous)

comportement durant le cycle de vie

- lorsque l'app est stoppée (par exemple l'écran s'éteint automatiquement après un certain délai), la couleur de fond du GridLayout change pour une couleur aléatoire
- si l'orientation de l'écran change, l'app est détruite puis est recréée comme nous l'avons vu. Trouvez une solution pour préserver l'état des *Views* (tous les *TextViews* avec leur contenu et leur couleur doivent être recréés)