Programmation des composants mobiles (Android)

Wieslaw Zielonka
zielonka@irif.fr

Une application Android

Les composants d'une application Android :

- **Activity** gère l'interface graphique
- **ContentProvider** gestion de données (fichiers, bases de données SQLite)
- **Service** implémente les opérations exécutées en arrière plan
- **Broadcast Receiver** écoute et réagit sur les événements système

Android studio

Création d’une application :

- `File -> Project` choisir le nom du projet,
- choisir Phone and Tablet et le minimum SDK,
- choisir Empty Activity,
- choisir le nom de l’activité.

Fichiers créés par Android studio

- le fichier Manifest :

app/manifests/AndroidManifest.xml

décrit tous les composants de l'application android. Par exemple pour chaque Activity une balise <activity android:name=".NomDeActivite">

- les fichiers sources kotlin par exemple

app/java/chemin_vers/Compteur.kt

- les fichiers xml layout dans le répertoire

app/res/layout/

qui définissent les interfaces graphiques de chaque activité. Pour la même activité nous pouvons avoir plusieurs fichiers layout (l'interface graphique peut dépendre de la taille de l'écran, de la position : verticale ou horizontale).

- les fichiers de ressources dans le répertoire

app/res/values/*

Modification de layout

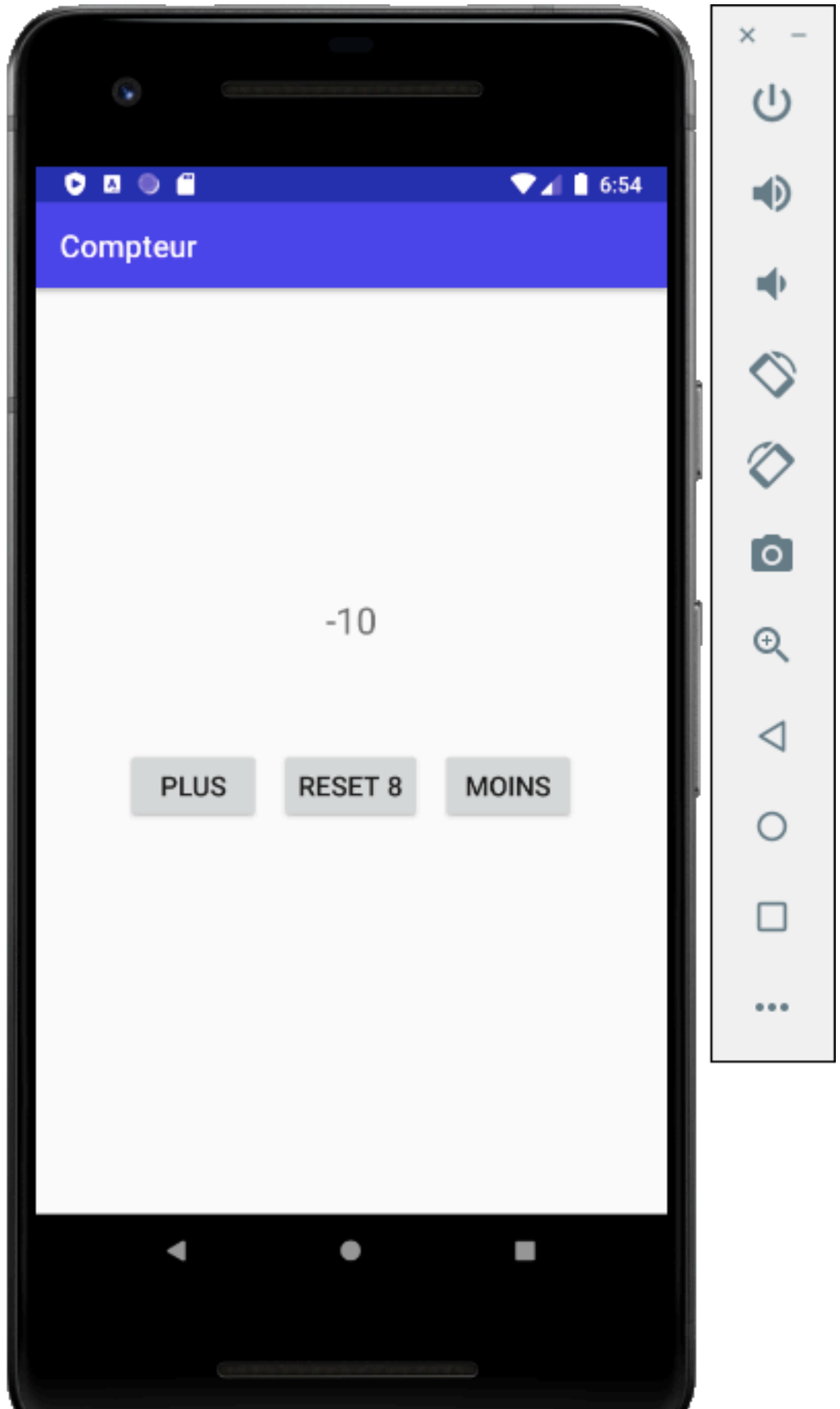
Le but est de modifier l'application vide créée par android.

Le but : l'application qui affiche un compteur entier (valeur 0 initialement) et trois boutons, le bouton plus, le bouton moins et le bouton reset..

Un click sur le bouton plus augmente la valeur du compteur, un click sur le bouton moins décrémente la valeur du compteur et reset remet le compteur à 0.

Première application Android

- Fichier layout
- Activity


6:54
Compteur
-10
PLUS
RESET 8
MOINS

Le fichier layout de l'activité

Au début nous allons fournir les fichiers layout. Il faudra au moins comprendre le contenu du fichier layout.

Pour chaque bouton le fichier layout contient une balise Button :

```
<Button
    android:id="@+id/reset"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingLeft="12dp"
    android:paddingRight="12dp"
    android:text="@string/reset"
    android:textSize="18sp" />
```

les attributs:

- text — le texte affiché dans le bouton, dans l'exemple le texte est dans les ressources

android:text="RESET"

pour indiquer directement le texte affiché dans le bouton (sans utilisation de ressources)

- `id` — identifiant de bouton, noter **@+id/** qui précède la valeur de l'identifiant
- `textSize` - la taille du texte
- `layout_width` et `layout_height` - largeur et hauteur, la valeur **`wrap_content`** indique la taille suffisante pour afficher le texte dans le bouton
- **paddingLeft, paddingRight** les marges

le fichier layout : TextView

TextView est un élément qui permet afficher un texte.

L'activité générée par android studio possède déjà un TextView, on change juste le texte affiché, la taille du texte et on ajoute un identifiant:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="0"
    android:textSize="35sp"
    android:layout_alignParentTop="true"
    android:id="@+id/text"
     ............/>
```

Les attributs les plus importants :

- text — le texte affiché dans le `TextView`
- textSize - la taille de police de caractères
- id — l'identifiant de TextView

Activity

```
class Compteur : AppCompatActivity() {  //classe dérivée de Activity
    var value: Int = 0   //la valeur de compteur

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        //créer l'interface graphique à partir de fichier layout
        // apt/res/layout/compteur.xml   (déserialisation de l'objet,
        // la création d'instances de tous les objets définis dans le layout)

        setContentView(R.layout.compteur)

        //récuperer les références vers les trois boutons et le TextView
        // text, plus, moins, reset : les identifiants de quatre objets
        // spécifiés dans les layout

        val txt :   TextView = findViewById(R.id.text)
        val plus :  Button   = findViewById(R.id.plus)
        val moins : Button   = findViewById(R.id.moins)
        val reset : Button   = findViewById(R.id.reset)

        //attacher les listeners aux boutons

    }
}
```

L'exécution commence dans la fonction onCreate() de l'activité principale.
Notez que les références vers les objets graphique peuvent être récupérées seulement après leurs création par setContentView

L’activité démarre par l’appel à la fonction `onCreate` :

```
override fun onCreate(savedInstanceState: Bundle?){  }
```

La méthode setContentView crée, à partir de la description donnée dans le fichier layout, tous les objets graphiques (widgets) qui forment l’interface graphique

```
setContentView(R.layout.compteur)
```

Le paramètre de `setContentView()` est le nom de fichier layout (précédé par R.layout).

Récupérer les références vers les objets qui constituent l'interface

La fonction

```
View  findViewById(int id)
```

retourne la référence vers un élément d'interface graphique. L'argument de la méthode : id de l'objet définie dans le fichier layout.

Donc si dans le fichier layout on a :

```
<Button
        . . .
        android:id=« @+id/plus" />
```

alors dans l'activité, dans onCreate()

```
val plus : Button = findViewById(R.id.plus)

val moins  = findViewById<Button>(R.id.moins)
```

Dans le fichier layout

```
android:id="@+id/text"
```

- @ indique une ressource
- + indique la première occurrence d'identifiant dans le fichier xml
- text – le nom de l’identifiant à choisir à votre guise

Associer un listener à un bouton

```
moins.setOnClickListener( listener );
```

`listener` – l'objet qui implémente l'interface

`View.OnClickListener`

qui contient une seule fonction :

```
fun onClick(v : View)
```

Associer un listener à un bouton

```
reset.setOnClickListener{  view -> value = 0
                                    txt.text="0"   }
```

Si une lambda expression possède un seul paramètre dont le type peut être "deviné" par le compilateur alors
ce paramètre peut-être omis :

```
reset.setOnClickListener{  value = 0
                           txt.text="0"   }
```

Mais quoi faire si le paramètre de lambda expression est utilisé dans la fonction ?

SAM : single abstract method interface

Associer un listener à un bouton

le bouton reset qui agit aussi bien sur le TextView (valeur de compteur) que sur le bouton reset lui-même :

```
reset.setOnClickListener{ view ->
    Toast.makeText(this, "reset", Toast.LENGTH_SHORT).show()
    reset.text = "RESET FROM ${value}")
    value = 0  //mettre compteur à 0
    txt.text="$value"
}
```

Après le reset le bouton RESET affiche la valeur du compteur avant le reset.

Le paramètre view n'est pas utilisé, on peut le supprimer:

```
reset.setOnClickListener{
    Toast.makeText(this, "reset", Toast.LENGTH_SHORT).show()
    reset.text = "RESET FROM $value"
    value = 0
    reset.text="$value"     //pourquoi ici pas la peine de faire
                           //le changement de
}
```

Associer un listener à un bouton

Même chose avec l'utilisation de paramètre de lambda :

```
reset.setOnClickListener{ view ->
    Toast.makeText(this, "reset", Toast.LENGTH_SHORT).show()
    (view as Button).text = "RESET FROM ${value}")
    value = 0  //mettre compteur à 0
    txt.text="$value"
}
```

Notez le retypage (view as Button). Le paramètre de lambda est une fonction de type View -> Unit donc le type du paramètre view est View.

Si une lambda expression possède un seul paramètre qui est omis dans la définition alors ce paramètre porte le nom **it** dans le corps de la fonction.

```
reset.setOnClickListener{
    Toast.makeText(this, "reset", Toast.LENGTH_SHORT).show()
    (it as Button).text = "RESET FROM $value"
    value = 0
    txt.text="$value"
}
```

cast (retypage) en Kotlin

```
( objet as Classs )   en Kotlin

( Classe ) objet      en java
```

Toast

```
Toast.makeText(this, "reset", Toast.LENGTH_SHORT).show()
```

`Toast` affiche un message `(ici` **`reset`**`)` en bas de l'écran pendant un court d'intervalle de temps

Sauvegarde de l'état de l'activité

Changement de contexte.

Quand on change la position de l'appareil(horizontale <—> verticale) l'activité courante est détruite et reconstituée avec l'appel à onCreate().

De même façon, si une activité A est exécutée et une autre activité B démarre, l'activité A est détruite (l'objet Activity de A est détruit). Si on revient avec le bouton BACK vers l'activité A, l'objet correspondant à l'activité A sera reconstruit et la méthode onCreate() de A exécutée.

Les valeurs de variables locales de A ne sont pas préservées, donc si on tourne l'appareil le compteur (TextView) sera reinitialisé à 0.

Question : et si le TextView devient la propriété de Activity ?

Les valeurs de propriétés ne sont pas mémorisées non plus pendant le changement de configuration, en général l'état de l'activité n'est pas préservé quand on tourne l'appareil (il y a une exception).

Préserver l’état de Activity avec un Bundle

Dans notre exemple nous voulons préserver la valeur du compteur.

Tout d'abord on sort le compteur de la méthode on onCreate() et on crée une propriété compteur de l'activité :

```
var value: Int = 0
```

On ajoute dans Actvity la méthode :

```
override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
    outState.putInt("key", value)
}
```

La méthode `onSaveInstanceState()` est appelée automatiquement par Android quand l’objet Activity est détruit (au changement de configuration).
Activity n'est pas détruit complètement, cette activité reste sur la pile des activités.

Il faut mettre dans le Bundle toutes les valeurs que vous voulez sauvegarder sous forme (clé, valeur) en utilisant différentes méthodes `put` de Bundle. Le paramètre clé est de type String.

Récupérer les valeurs sauvegardées dans Bundle dans la méthode onCreate()

Le `Bundle` sauvegardé dans `onSaveInstanceState()` devient le paramètre de `onCreate()` quand l'activité redémarre. On ajoute le code pour récupérer la valeur de compteur:

```
class Compteur : AppCompatActivity() {
    var value: Int = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContentView(R.layout.compteur)
        val txt : TextView = findViewById(R.id.text)
        val plus : Button = findViewById(R.id.plus)
        val moins : Button = findViewById(R.id.moins)
        val reset : Button = findViewById(R.id.reset)

        //si saveInstanceState est null alors value = 0
        //sinon value = la valeur sauvegardée dans savedInstanceState : Bundle

        value = savedInstanceState?.getInt("key") ?: 0
        txt.setText("$value")
```

Pour chaque méthode `put()` le `Bundle` possède une méthode `get()` qui permet de récupérer la valeur associée à une clé.
Quand l'activité démarre pour la première fois le paramètre Bundle? de onCreate() est nulle.

L'activité complète

```
package fr.irif.zielonka.compteur
/* les imports  */

class Compteur : AppCompatActivity() {
    var value: Int = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.compteur)
        val txt : TextView = findViewById(R.id.text)
        val plus : Button = findViewById(R.id.plus)
        val moins : Button = findViewById(R.id.moins)
        val reset : Button = findViewById(R.id.reset)

        value = savedInstanceState?.getInt("key") ?: 0
        txt.setText(value.toString())

plus.setOnClickListener {
    Toast.makeText(this, "plus", Toast.LENGTH_SHORT)
        .show()
    value++
    txt.setText(value.toString())
}

moins.setOnClickListener{
        Toast.makeText( this@Compteur, "moins", Toast.LENGTH_SHORT)
          .show()
        value--
        txt.setText("$value")
    }
})
```

l'activité complète (suite)

```
        //reset.setOnClickListener{ v : View ->  txt.text="0" }
        reset.setOnClickListener{
            Toast.makeText(this, "reset", Toast.LENGTH_SHORT).show()

            reset.setText("reset ${value}")
            value = 0
            txt.text=value.toString()
        }

    }//end onCreate()

    override fun onSaveInstanceState(outState: Bundle) {
            super.onSaveInstanceState(outState)
            outState.putInt("key", value)
        }
    }
```

Initialisation de propriétés

```
class Compteur : AppCompatActivity() {

    val txt : TextView = findViewById(R.id.text) //incorrect

    var value: Int = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.compteur)

        val plus : Button = findViewById(R.id.plus)
        val moins : Button = findViewById(R.id.moins)
        val reset : Button = findViewById(R.id.reset)
```

Ce code est incorrect. Initialisation des propriétés de Compteur se fait avant l'appel à onCreate(), mais les éléments graphiques (view) sont créés par setContentView() dans onCreate().

Donc findViewById() ne pourra pas retourner le TextView qui n'existe pas encore au moment de l'appel.

Initialisation de propriétés

```
class Compteur : AppCompatActivity() {

    var txt : TextView //incorrect

    var value: Int = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.compteur)

        txt = findViewById(R.id.text)
        val plus : Button = findViewById(R.id.plus)
        val moins : Button = findViewById(R.id.moins)
        val reset : Button = findViewById(R.id.reset)
```

Hélas, une propriété non-nullable doit être initialisée à la création. Donc ce fragment de code est incorrect.

La définition suivante de la propriété txt est correct mais le fait que cette propriété soit nullable alourdira le code.

```
var txt : TextView?
```

Initialisation de propriétés

```
class Compteur : AppCompatActivity() {

    //correct mais embêtant,
    //il faut chaque fois vérifier si txt n'est pas null

    var txt : TextView?

    var value: Int = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.compteur)

        txt = findViewById(R.id.text)
        val plus : Button = findViewById(R.id.plus)
        val moins : Button = findViewById(R.id.moins)
        val reset : Button = findViewById(R.id.reset)
```

Ici txt est initialisé avec la valeur null. Nous oblige à vérifier chaque fois si la valeur est null ou non, très très embêtant

Initialisation paresseuse d'une propriété mutable

```
class Compteur : AppCompatActivity() {

    latinit var txt : TextView
    var value: Int = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.compteur)

        txt =   findViewById(R.id.txt)
        val plus : Button = findViewById(R.id.plus)
        val moins : Button = findViewById(R.id.moins)
        val reset : Button = findViewById(R.id.reset)
```

latinit indique que la propriété sera initialisé plus tard. La propriété latinit est toujours mutable.

On peut faire mieux.

Initialisation paresseuse d'une propriété non-mutable

```
class Compteur : AppCompatActivity() {

    val txt : TextView by lazy{ findViewById(R.id.text) as TextView }

    var value: Int = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.compteur)


        val plus : Button = findViewById(R.id.plus)  */
        val moins : Button = findViewById(R.id.moins)
        val reset : Button = findViewById(R.id.reset)
```

txt sera initialisé avec la valeur retournée par findViewById() mais l'appel à findViewById(R.id.text) sera exécuté seulement quand, pour la première fois pendant l'exécution, kotlin aura besoin d'une référence à txt.

Installer le même listener pour plusieurs boutons

```
val txt : TextView = findViewById(R.id.text)
val plus : Button = findViewById(R.id.plus)
val moins : Button = findViewById(R.id.moins)
val reset : Button = findViewById(R.id.reset)
```

//définir un objet qui implémente l'interface View.OnClickListener

```
val listener = View.OnClickListener{

        when(it){
            plus -> { value++}
            moins -> { value--}
            reset -> { value=0}
        }
        txt.text="$value"
    }
}

plus.setOnClickListener(listener)
moins.setOnClickListener(listener)
reset.setOnClickListener(listener)
```

Installer le même listener pour plusieurs boutons

```
val txt : TextView = findViewById(R.id.text)
val plus : Button = findViewById(R.id.plus)
val moins : Button = findViewById(R.id.moins)
val reset : Button = findViewById(R.id.reset)
```

//définir un objet qui implémente l'interface View.OnClickListener

```
val listener = {  view : View? -> Unit

        when(view){
              plus -> { value++}
              moins -> { value--}
                  reset -> { value=0}
            }
            txt.text="$value"
        }
}

plus.setOnClickListener(listener)
moins.setOnClickListener(listener)
reset.setOnClickListener(listener)

La méthode onClick est de type       View? -> Unit
```