Laboratoire 2

Activités, Intentions et filtres d'intentions

Objectifs d'apprentissage

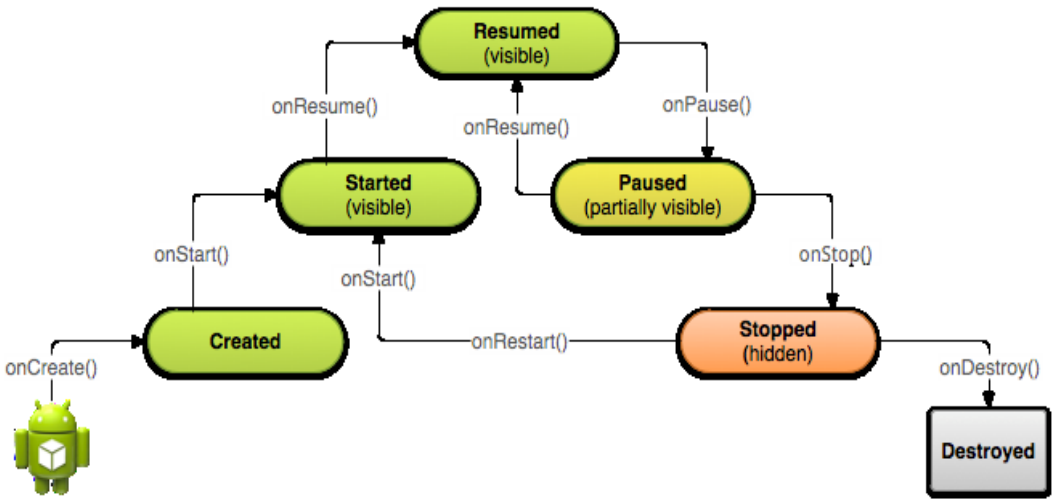
- Comprendre le cycle de vie d'une activité
- Sauvegarde et restauration de l'état d'une activité
- Les intentions explicites : lancement et échanges d'information entre activités.
- Les intentions implicites
- Les filtres d'intention
- Déclarations des intentions implicites dans le fichier manifest

Important

Utiliser le fichier ressource strings.xml pour tous les textes utilisés avec vos widgets (bouton, textView,…)

1- Cycle de vie d'une activité

Le système Android exécute le code des fonctions de rappel (Callback) selon l'état d'une activité (voir le schéma suivant tiré du site Android)



Il faut programmer certaines fonctions de rappel pour assurer le bon fonctionnement de l'application.

- **Resumed** : l'activité est affichée et utilisée (interaction avec l'utilisateur). On parle aussi de l'état d'exécution (running).
- **Paused** : l'activité est visible mais n'a pas le focus (quand on affiche une boite de dialogue par exemple). Elle passe toujours par cet état avant l'état *stopped.* C'est ici qu'il faut libérer certaines ressources (la caméra par exemple) qui pourraient être réallouées dans l'état *resumed.*
- **Stopped** : l'activité est complètement cachée. Les informations sur son état sont conservées.
- **Created** et **Started** : ces deux états sont transitoires. Après *onCreate(),* le système appelle systématiquement les méthodes *onStart()* et *onResume().* La méthode *onCreate()* est évoquée une seule fois durant le cycle de vie d'une activité. C'est donc dans cette méthode qu'on définit l'interface graphique et on initialise certaines variables.
- **Destroyed**: l'activité est détruite dans les cas suivants :
    - Par l'utilisateur à l'aide du bouton *retour-arrière*
    - Par elle-même en évoquant la méthode *finish()*
    - Par le système si l'activité est restée longtemps dans l'état *stopped*
    - Par le système si l'activité qui a le focus requiert plus de ressources
    - Par le système lors du changement d'orientation de l'écran (dans ce cas, elle est aussitôt recréée)

1.1 Programmation des fonctions de rappel et utilisation de la classe Log

- Créer un nouveau projet ***File/new/new Project*** avec comme nom d'application **Laboratoire 2.**
- Choisir l'onglet *Design*
- Vous allez utiliser un autre Layout qui s'appelle ***LinearLayout*** à la place du Layout par défaut qui est le ***ConstraintLayout***.
  Faites un clic droit sur **ConstraintLayout** dans le panneau "Component tree".
- Cliquez sur "Convert View" et choisissez l'option **LinearLayout** et cliquez sur le bouton ***Apply***. Remarquer que le texView change automatiquement de place en convertissant le **Layout**. Ce type de Layout aligne les éléments de droite à gauche (orientation horizontale) ou de haut en bas (orientation verticale).
- Supprimer le ***textView*** qui est créé par défaut avec l'activité (Sélectionnez le ***textView*** et appuyez sur la touche Suppr.)

- Changez l'orientation du LinearLayout en ***vertical*** en faisant un clic droit dessus et choisissez ***Convert orientation to vertical***.
  Dans ce cas, les composants seront placés l'un au-dessous de l'autre.
- Donnez la valeur *layoutBase* à la propriété *id* du *Layout.* Vérifier que ses propriétés *width* et *height* ont pour valeur *match_parent*. De cette façon, le *layout* occupera tout l'écran.
- Ajouter une zone de texte (Large Text) comme dans l'écran suivant sans oublier d'utiliser le fichier ressource strings.xml pour le contenu. Attention à l'apostrophe dans le fichier strings.xml.

Ajouter le code suivant à l'activité *MainActivity.kt* après la fonction onCreate.

```
override fun onStart() {
    super.onStart()
    Log.i("MainActivity ", "dans ${object {}.javaClass.enclosingMethod.name}")
}

override fun onResume() {
    super.onResume()
    Log.i("MainActivity ", "dans ${object {}.javaClass.enclosingMethod.name}")
}

override fun onPause() {
    super.onPause()
    Log.i("MainActivity ", "dans ${object {}.javaClass.enclosingMethod.name}")
}

override fun onStop() {
    super.onStop()
    Log.i("MainActivity ", "dans ${object {}.javaClass.enclosingMethod.name}")
}

override fun onDestroy() {
    super.onDestroy()
    Log.i("MainActivity ", "dans ${object {}.javaClass.enclosingMethod.name}")
}

override fun onRestart() {
    super.onRestart()
    Log.i("MainActivity ", "dans ${object {}.javaClass.enclosingMethod.name}")
}
```

Ajouter le code suivant à l'activité *MainActivity.kt* à la fin de la fonction onCreate

```
Log.i("MainActivity ", "dans ${object {}.javaClass.enclosingMethod.name}")
```

Noter l'utilisation de l'instruction Log.i (équivalent de la fonction println de Kotlin).

Il existe 5 options d'affichage dans la classe Log :

- v : verbeux, affiche toutes les informations
- d : affiche les informations en mode débogage
- e : affiche les informations en mode erreur
- i : affiche les informations en mode info
- w : affiche les informations en mode warning

Pour utiliser la classe Log, choisir l'onglet *LogCat* dans le panneau du bas de l'IDE.

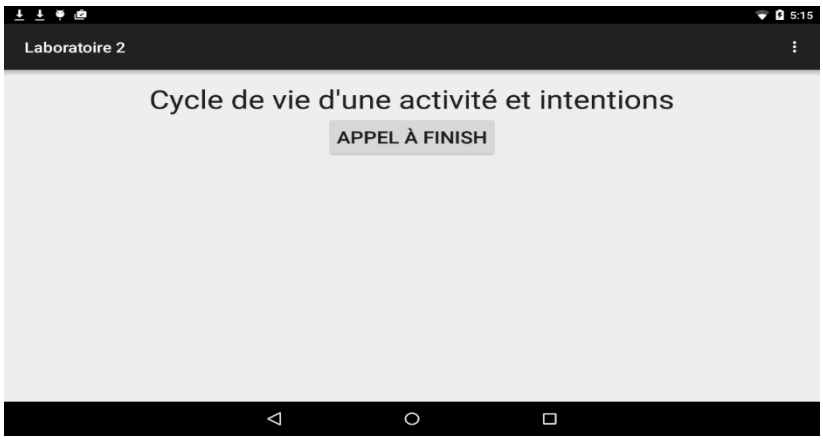
Nous pouvons aussi restreindre les informations affichées dans le LogCat en utilisant un filtre. Voici les étapes :

- Dans la liste show *only selected application*, choisir *Edit Filter Configuration*
- Donner un nom à votre filtre
- Entrer *MainActivity* dans la zone *Log Tag*
- Dans la liste déroulante by *Log Level,* choisir *info*

1.2 Test des fonctions de rappel

Pour observer les différents états de l'activité, cliquer sur le nom du filtre choisi dans la liste

- Démarrer votre application. Observer le passage automatique par *onCreate(), onStart()* et *onResume()*
- Naviguer sur une autre application (utiliser *Home* par exemple). Noter les appels à *onPause()* et *onStop()*.
- Relancer l'application en utilisant l'icône des applications récentes. L'activité passe par *onRestart(), onStart()* et *onResume()*.
- Faire un retour arrière. Noter que dans ce cas l'activité est détruite. Elle passe à l'état *Destroyed*
- Relancer l'application. À l'aide de l'icône des applications récentes. Noter que l'activité passe onCreate()() *onStart()* et on *Resume()*
- Changer l'orientation de l'écran. Observer que l'activité est détruite puis recréée automatiquement (onCreate()….)
- Ajouter un bouton *APPEL À FINISH* à votre interface comme suit:



➢ Programmez l'écouteur d'événement du bouton dans la méthode *onCreate()* de *MainActivity* dont le rôle est d'appeler la fonction *finish().* Pour ce faire, ajoutez la ligne suivante à la fin de la fonction onCreate.

```
btnFinish.setOnClickListener { finish() }
```

➢ Exécutez. Cliquez sur le bouton APPEL À *FINISH*. Noter que l'activité passe à l'état *destroyed* et disparaît de l'écran, ce qui termine l'application.

1.3 Sauvegarde et restauration de l'état d'une activité

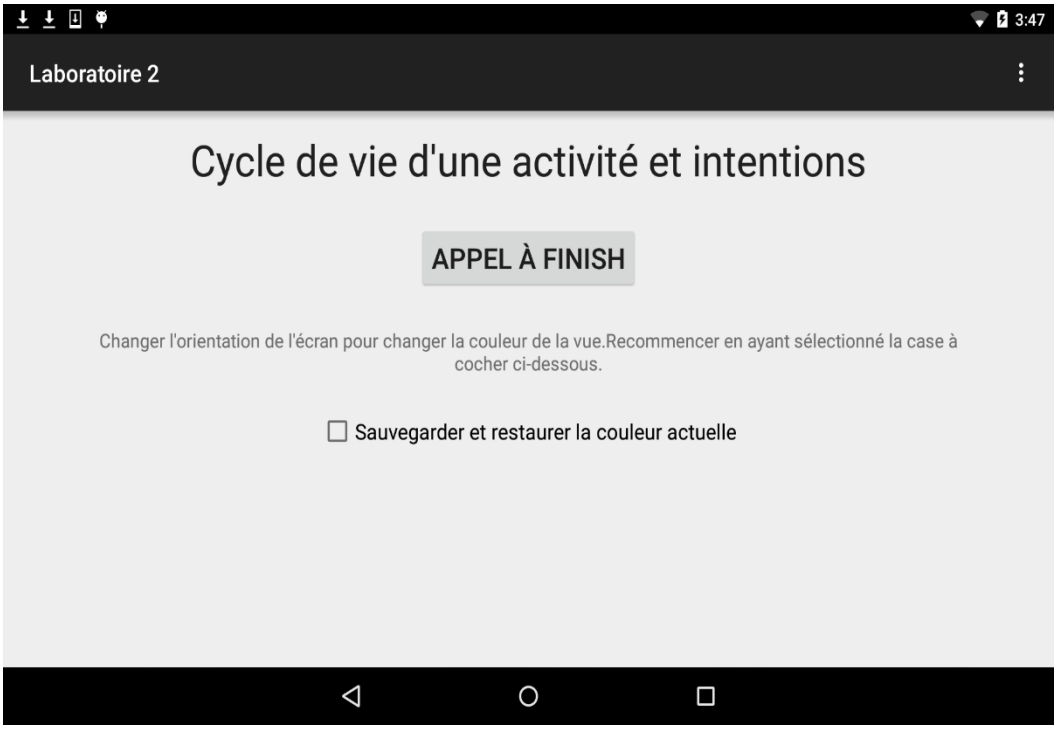
Lorsqu'une activité est détruite à cause d'un changement d'orientation de l'écran ou parce que le système a besoin de ressources, le système recrée l'interface graphique et son contenu. Il vous donne la possibilité de sauvegarder les informations nécessaires dans un objet *Bundle* semblable à une *Hashtable*. Si d'autres informations sont importantes pour la reprise de l'activité, il faudrait les ajouter à cet objet. Pour ce faire, le SDK fournit deux autres méthodes de rappel : ***onSaveInstanceState*** et ***onRestoreInstanceState***. La méthode ***onSaveInstanceState*** est appelée avant la méthode ***onStop***. ***onRestoreInstanceState*** est appelée après la méthode ***onStart*** et avant la méthode ***onResume.***

- Ajouter le code suivant à la fin de la classe *MainActivity*

```
override fun onSaveInstanceState(outState: Bundle, outPersistentState: PersistableBundle) {
    super.onSaveInstanceState(outState, outPersistentState)
    Log.i("MainActivity ", "dans ${object {}.javaClass.enclosingMethod.name}")
    Toast.makeText(this, "dans ${object {}.javaClass.enclosingMethod.name}",Toast.LENGTH_LONG).show()
}

override fun onRestoreInstanceState(savedInstanceState: Bundle) {
    super.onRestoreInstanceState(savedInstanceState)
    Log.i("MainActivity ", "dans ${object {}.javaClass.enclosingMethod.name}")
    Toast.makeText(this, "dans ${object {}.javaClass.enclosingMethod.name}",Toast.LENGTH_LONG).show()
}
```

- Effacer les messages de votre Logcat (clic droit/clear LogCat)
- Exécuter votre programme
- Naviguer vers une autre application, observer le passage par la méthode *onSaveInstanceState().*
- Relancer votre application, observer les fonctions appelées.
- Changer l'orientation de l'écran. Noter la présence de la méthode *onSaveInstanceState()* et *onRestoreInstanceState().*
- Faire un retour arrière. Observer les fonctions appelées.
- Relancer l'application et observer à nouveau les fonctions appelées.
- Pour mieux saisir l'intérêt de ces deux méthodes. Compléter votre interface graphique comme suit :



- Ajouter les variables de classe suivantes à *MainActivity*

```
private var CLE_COULEUR:String = "COULEUR"
private var mCouleur:Int = 0
```

- Ajoutez la ligne suivante à la fin de la fonction onCreate :

```
mCouleur = Random(System.currentTimeMillis()).nextInt()
```

- Compléter la méthode *onResume()* avec:

```
layoutBase.setBackgroundColor(mCouleur)
```

- Compléter la méthode ***onSaveInstanceState(Bundle outState)*** avec :

```
if(checkBox.isChecked){
    outState.putInt(CLE_COULEUR, (layoutBase.background as ColorDrawable).color)
}
```

- Compléter la méthode ***onRestoreInstanceState(Bundle SavedInstanceState)*** avec :

```
if(checkBox.isChecked) {
    mCouleur = savedInstanceState.getInt(CLE_COULEUR)
}
```

- Exécuter et bien comprendre l'importance de programmer ces méthodes.

2- Les intentions

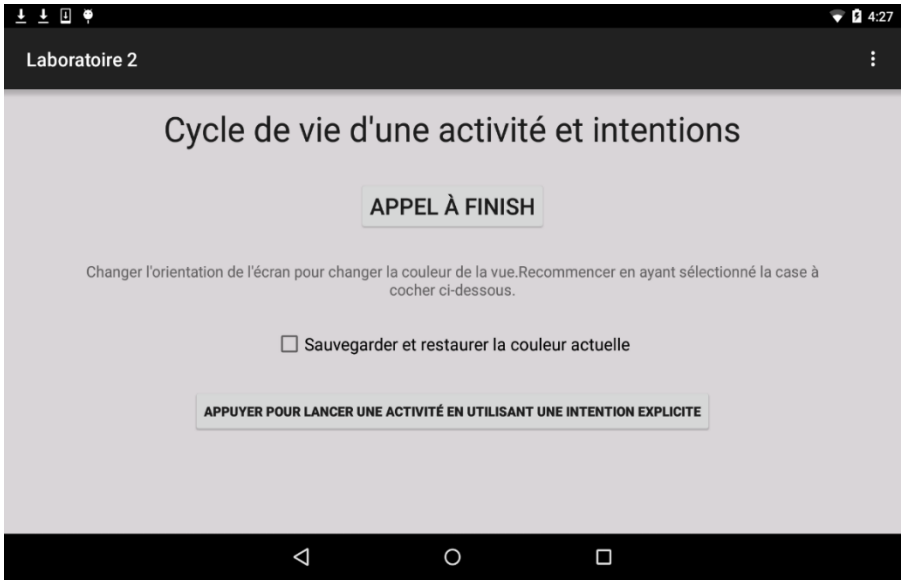
Les intentions permettent la communication entre les différents composants.

2.1 Intention explicite- Cas simple

Une intention explicite désigne précisément le composant auquel elle est destinée.

Les intentions explicites sont essentiellement utilisées pour lancer des composants applicatifs à l'intérieur d'une même application.

- Ajouter un bouton (Id : btnExplicite) à votre application de manière à avoir ce qui suit :



Dans cette partie, nous allons créer une nouvelle activité sans utiliser les automatismes du SDK.

- Dans l'ensemble le package ca.qc.*cgodin.laboratoire2* ajouter une nouvelle classe ***DestinataireActivity***, sous-classe de la classe AppCompat*Activity* (: AppCompatActivity)
- Créer une vue pour cette activité. Dans layout->new->Layout Resource File, donner le nom *activity_destinataire* à votre fichier.
- Ajouter un composant *textView* au centre de *activity_destinataire*. Le composant *textView* doit afficher le message « *Ceci est l'activité destinataire. Appuyer sur la touche retour pour revenir à l'activité appelante* »
- Pour associer le layout précédent à votre activité, modifiez le code de votre activité *DestinataireActivity* comme suit après

```
package ca.qc.cgodin.laboratoire2

import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity


class DestinataireActivity : AppCompatActivity() {
    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_destinataire)
    }
}
```

- Ajouter le code suivant à *la fin de la fonction onCreate*

```
btnExplicit.setOnClickListener(this)
```

Vous allez recevoir un message d'erreur. Vous devez le corriger en implémentant l'interface View.OnClickListener.

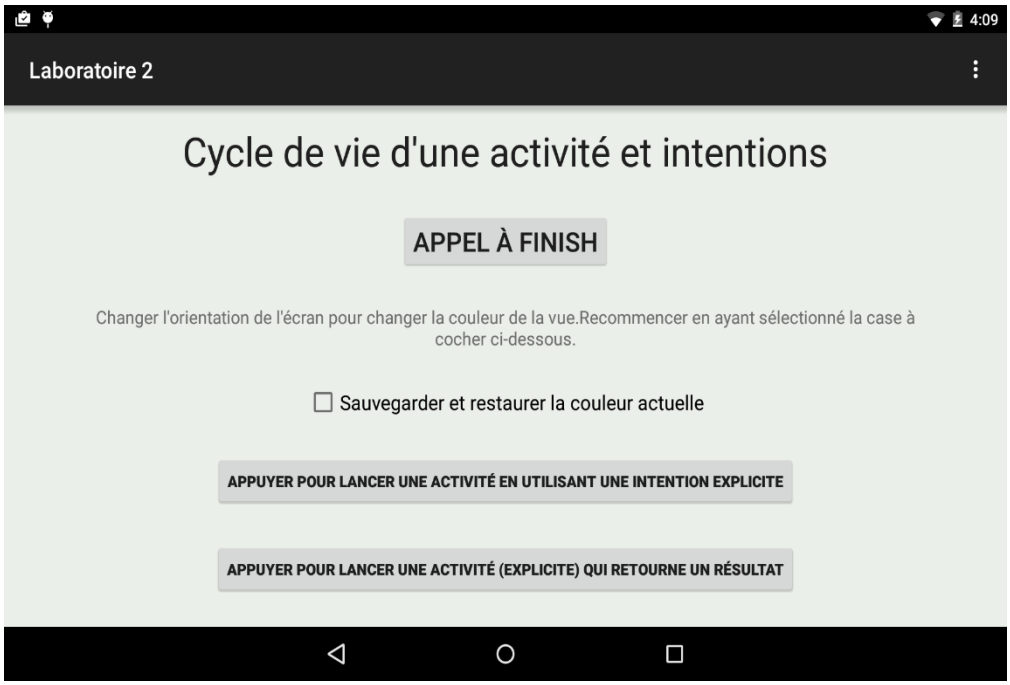
```
class MainActivity : AppCompatActivity(), View.OnClickListener {
```

- Exécuter votre application. Le programme plante.
  Pour régler le problème, il faut que la nouvelle activité soit aussi déclarée dans le fichier *AndroidManifest* comme suit :
  <activity android:name=".DestinataireActivity"/>
- Exécuter à nouveau votre application.

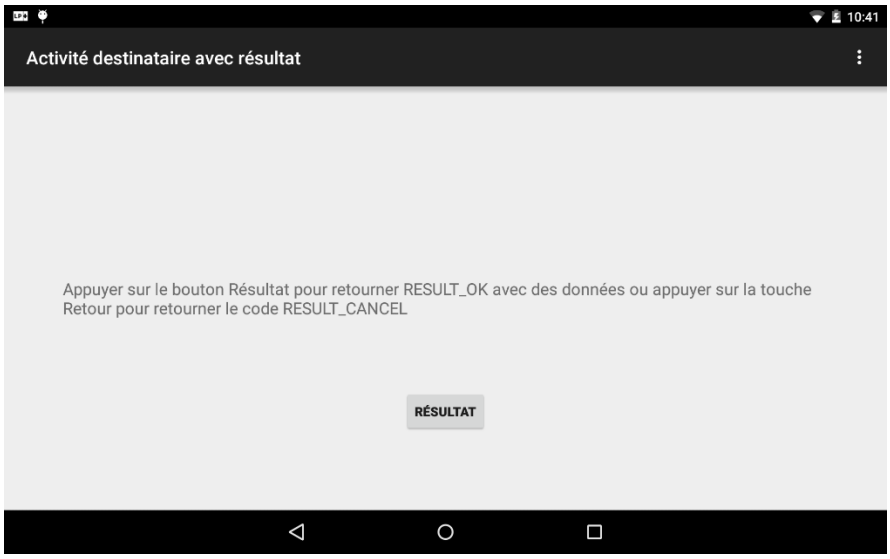
2.2 Intention Explicite- Cas avec retour de code résultat

- Dans ce cas l'activité appelante attend des résultats de l'activité destinataire.

- Ajouter un bouton (Id: btnRes) à *activity_main.xml* de manière à avoir ce qui suit :



- Dans l'ensemble de classes *ca.qc.cgodin,* ajouter une activité *DestinataireResActivity comme suit :*
  new->Activity->EmptyActivity.
- Dans ce cas, vous remarquerez qu'une vue *activity_destinataire_res.xml* est automatiquement associée à votre nouvelle activité. Voir aussi le *setContentView* dans la méthode *onCreate* de *DestinataireResActivity.* Vous allez aussi remarquer que dans ce cas l'activité est automatiquement déclarée dans le fichier *AndroidManifest.*
- *Glisser* un *textView* au centre de *activity_destinataire_res* ainsi qu'un bouton (*btnResultat*) de commande comme suit :



Noter que pour avoir le titre ***Activité destinataire avec résultat***, il faut ajouter, dans le fichier ***strings.xml***, la valeur de la variable *title_activity_destinataire_res* que vous devez définir comme suit dans le fichier *AndroidManifest.xml*. L'activité existe déjà dans le fichier AndroidManifest. Il faut ajouter la ligne en gras qui vous permet d'avoir un titre pour l'activité.

```
<activity
        android:name=".DestinataireResActivity"
        android:label="@string/title_activity_destinataire_res" >
 </activity>
```

- Compléter le code de la classe *DestinataireResActivity* comme suit :

```
import android.app.Activity
import android.content.Intent
import android.os.Bundle
import android.view.View
import android.widget.Button
import com.google.android.material.snackbar.Snackbar
import androidx.appcompat.app.AppCompatActivity;

import kotlinx.android.synthetic.main.activity_destinataire_res
.*

class DestinataireResActivity : AppCompatActivity() {

    companion object{
        val CLE1 = "cle1"
        val CLE2 = "cle2"
    }
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_destinataire_res)
        setResult(Activity.RESULT_CANCELED)
        val btnRes = findViewById<Button>(R.id.return_detinatai
re_btn)

        btnRes.setOnClickListener {
                val donnees = Intent()
                donnees.putExtra(CLE1, "allo")
                donnees.putExtra(CLE2, true)
                setResult(RESULT_OK, donnees)
                finish()
            }
    }

}
```

Noter bien l'utilisation des méthodes ***setResult et putExtra.***
putExtra sur un objet ***Intent*** permet de stocker de l'information qui pourra être récupérée par l'activité appelante.