Laboratoire 2

Activités, Intentions et filtres d'intentions

Objectifs d'apprentissage

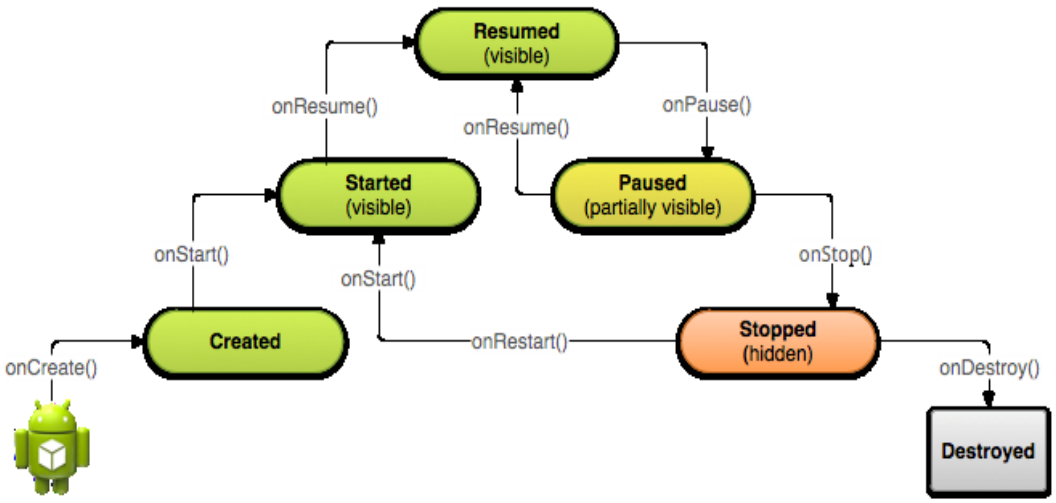
- Comprendre le cycle de vie d'une activité
- Sauvegarde et restauration de l'état d'une activité
- Les intentions explicites : lancement et échanges d'information entre activités.
- Les intentions implicites
- Les filtres d'intention
- Déclarations des intentions implicites dans le fichier manifest

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 position 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. Remarquez que le texte est centré horizontalement.

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.

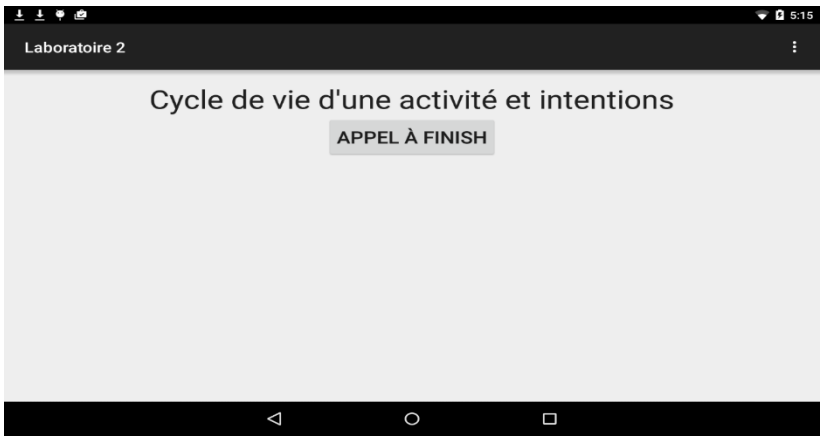
Nous pouvons aussi restreindre les informations affichées dans le LogCat en utilisant un filtre. Voici les étapes :

- 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 par onPause, onStop, onDestroy. L'activité 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

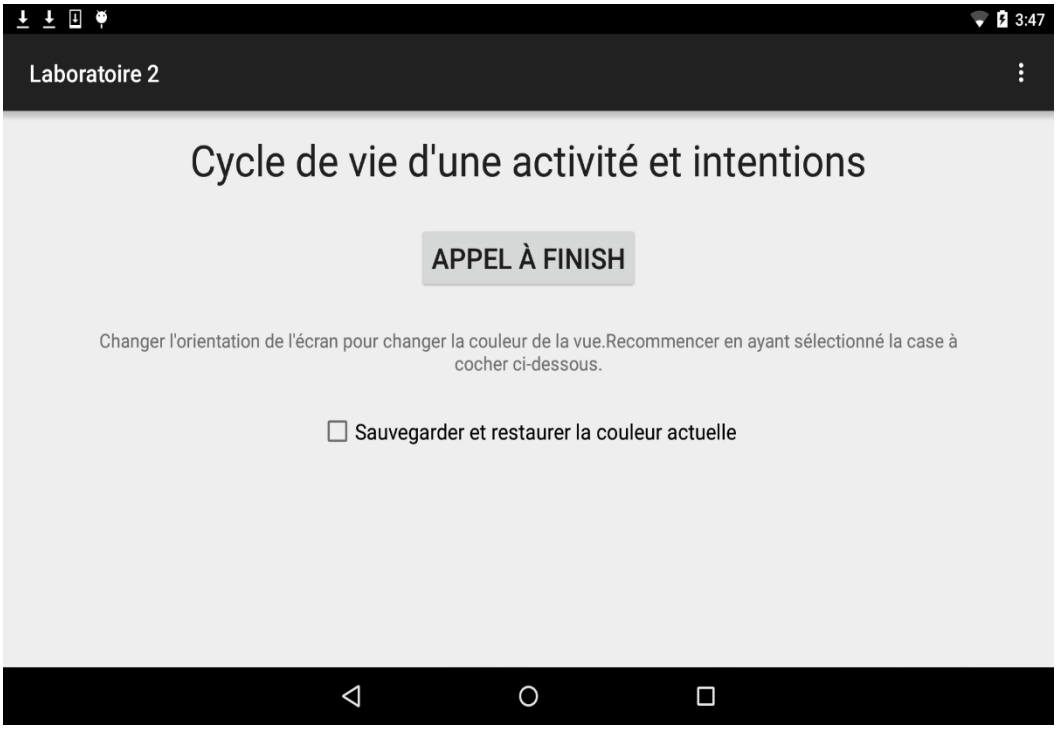
Lorsqu'une activité est détruite à cause d'un changement d'orientation de l'écran ou parce que le système a besoin de ressources, le système recrée l'interface graphique et son contenu. Il vous donne la possibilité de sauvegarder les informations nécessaires dans un objet *Bundle* semblable à une *Hashtable*. 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.* Importez les classes nécessaires.

```
override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
    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 (le checkbox ayant pour ID checkbox):



- 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 ces 2 ligne à la fin(le checkbox ayant pour ID checkbox):

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

- Compléter la méthode ***onRestoreInstanceState(Bundle SavedInstanceState)*** avec ces 2 lignes à la fin(le checkbox ayant pour ID checkbox):

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

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

2- Les intentions

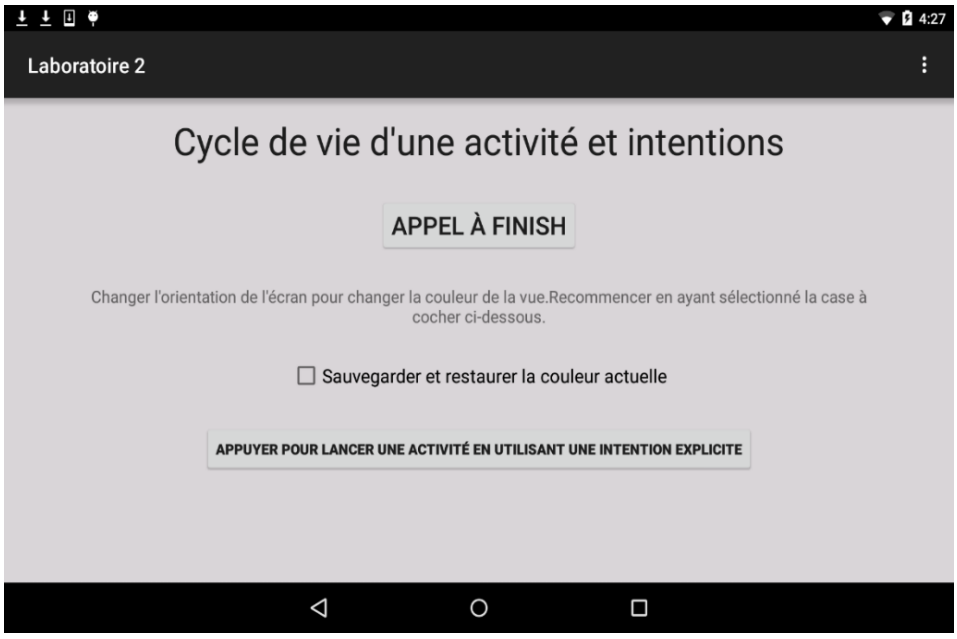
Les intentions permettent la communication entre les différents composants.

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 : btnExplicit) 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.

- Ajoutez une nouvelle classe ***DestinataireActivity*** au package ca.qc.*cgodin.laboratoire2*, 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* de la classe MainActivity

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

Vous allez recevoir un message d'erreur. Vous devez le corriger en implémentant l'interface View.OnClickListener, ajoutant aussi une fonction onClick à votre classe MainActivity.

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

- Modifiez la fonction onClick qui a été ajoutée en implémentant View.OnClickListener comme suit:

```
override fun onClick(view: View?) {
    if (view === btnExplicit) {
        val intent = Intent(this@MainActivity, DestinataireActivity::class.java)
        startActivity(intent)
    }
}
```

- Exécuter votre application. Le programme plante quand vous cliquez sur le bouton btnExplicit. Vérifiez le message d'erreur dans Logcat.

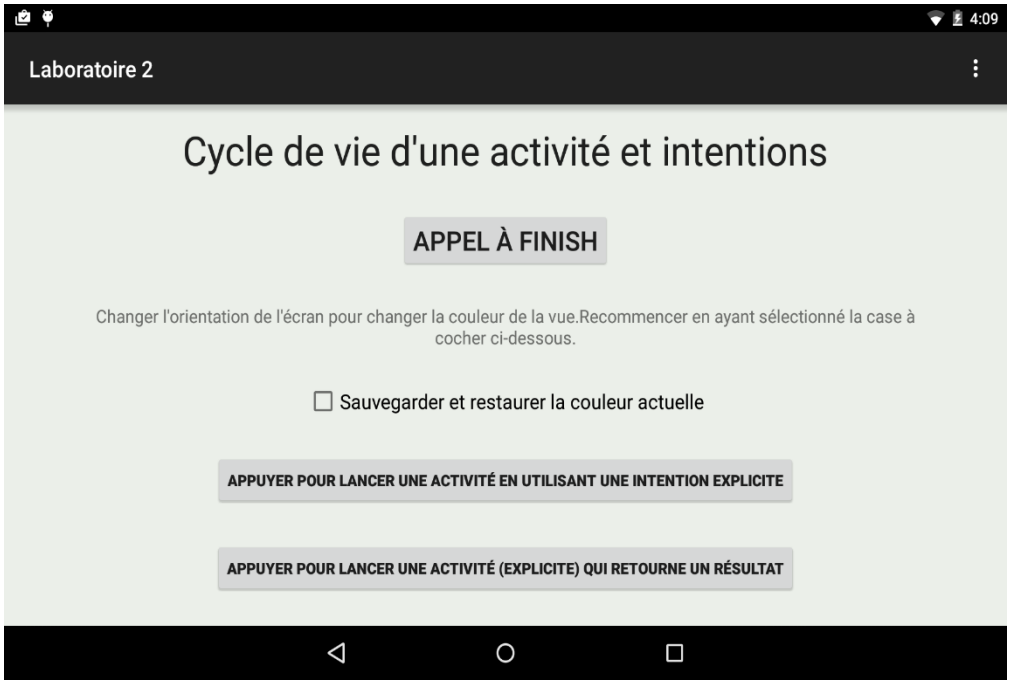
  Pour régler le problème, il faut que la nouvelle activité soit aussi déclarée dans le fichier *AndroidManifest,* ce que vous devez faire comme suit :

```
<activity android:name=".DestinataireActivity" />
```

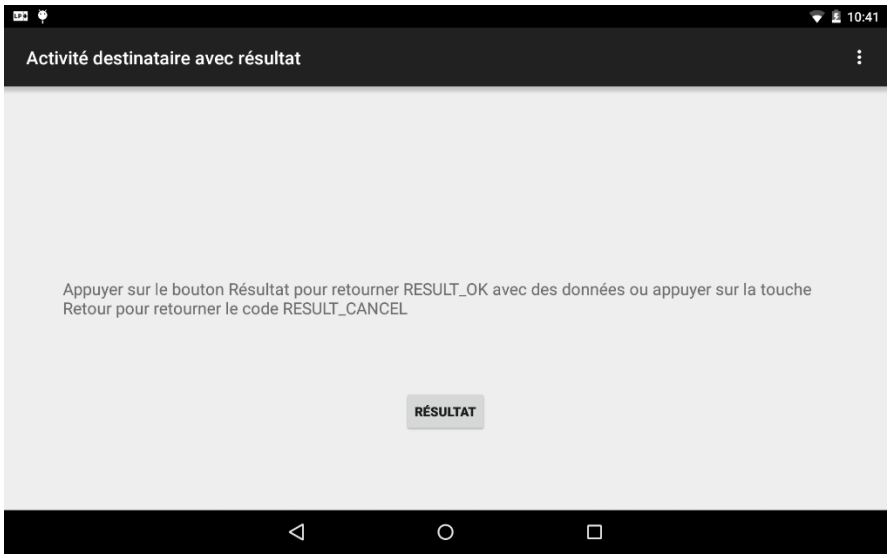
- Exécuter à nouveau votre application. Appuyez sur le bouton btnExplicit pour afficher l'activité destinataire. Vous devez voir ce qui suit:

2.2 Intention Explicite- Cas avec retour de code résultat

- Dans ce cas d'intention explicite, 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 :



- Faites un clic droit sur le package *ca.qc.cgodin.laboratoire2,* créez une nouvelle activité *DestinataireResActivity en cliquant sur New->Activity->EmptyActivity.*
- Dans ce cas, vous remarquerez qu'une vue *activity_destinataire_res.xml* est automatiquement associée à votre nouvelle activité comme il en est dans la création de votre MainActivity de départ. 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 (ID: *btnResultat*) 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>
```

- Ajoutez la ligne suivant à la fin de la fonction onCreate de la classe MainActivity

```
btnRes.setOnClickListener (this)
```

Modifiez la function onClick de la classe MainActivity comme suit:

```
override fun onClick(view: View?) {
    when(view){
        btnExplicit ->{
            val intent = Intent(this@MainActivity, DestinataireActivity::class.java)
            startActivity(intent)
        }
        btnRes ->{
            val intent = Intent(this, DestinataireResActivity::class.java)
            startActivityForResult(intent, 1)
        }
    }
}
```

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

```
package ca.qc.cgodin.laboratoire2

import android.app.Activity
import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import kotlinx.android.synthetic.main.activity_main.*

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)

        btnResultat.setOnClickListener {
            val donnees = Intent()
            donnees.putExtra(CLE1, "allo")
            donnees.putExtra(CLE2, true)
            setResult(Activity.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.

- Compléter l'activité appelante *MainActivity* comme suit :

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    Log.i("MainActivity ", "dans ${object {}.javaClass.enclosingMethod.name}")
    when (requestCode) {
        1 -> {
            val resTxt = if (resultCode == Activity.RESULT_OK) {
                "RESULT_OK " + DestinataireResActivity.CLE1 + "= " +
data?.getStringExtra(DestinataireResActivity.CLE1) + " " +
                        DestinataireResActivity.CLE2 + "= " +
data?.getBooleanExtra(DestinataireResActivity.CLE2, false)
            } else
                "RESULT_CANCELED"
            Toast.makeText(this, resTxt, Toast.LENGTH_LONG).show()
        }
        else -> super.onActivityResult(requestCode, resultCode, data)
    }
}
```

startActivityForResult est utilisée pour lancer l'activité destinataire.

La fonction de rappel ***protected void onActivityResult(int requestCode, int resultCode, Intent data)*** est exécutée juste après ***stop()*** et juste avant le ***restart()*** de l'activité appelante. Vérifier à l'aide de ***Log.i("MainActivity","dans onActivityResult");***

- Relancez l'application et tester en cliquant sur le bouton Résultat puis sur le retour arrière à partir de l'activité destinataire.

2.3 Intention Implicite

Contrairement à l'intention explicite qui désigne le composant destinataire, l'intention implicite indique l'action à réaliser. C'est le système qui se charge de trouver les composants destinataires capables d'exécuter cette action. S'il en existe plusieurs, le système demandera à l'utilisateur d'en choisir un.

- Compléter le *layout activity_main* en ajoutant un *textView*, un *LinearLayout horizontal* contenant trois boutons comme le montre la figure suivante :

- Utiliser la propriété *layout_weigth* des boutons pour leur attribuer le même espace dans le *layout* qui les contient (mettre à 1 cette valeur).
- Nous allons utiliser une autre façon pour programmer les clics sur les boutons.

  Donnez la valeur *onClickButtonACTION1* à la propriété *On Click* du bouton *ACTION1.*

  Copiez le code suivant dans *MainActivity*

```
fun onClickButtonAction1(view: View) {
    val intent = Intent("ca.qc.cgodin.laboratoire2.ACTION1")
    intent.addCategory("ca.qc.cgodin.laboratoire2.CATEGORIE1")
    startActivity(intent)
}
```

  Ce code permet de **créer une intention implicite**. Noter qu'il est important de préciser l'action. Dans le cas d'une action définie par l'utilisateur, cette-ci doit être préfixée par le nom du package du projet. La catégorie fait aussi partie des critères utilisés par le système lors du choix de l'intention à exécuter.
- Exécuter votre application. Cliquer sur le bouton ACTION1. Elle plante (Vérifie le message d'erreur dans Logcat). Le système n'a pas trouvé de composant applicatif susceptible de réaliser l'action que vous avez demandée. Pour résoudre ce problème, il faut qu'il y ait un composant applicatif qui déclare au système, par le biais de *filtres d'intentions,* le type d'intention implicite auquel il peut répondre. C'est ce que nous ferons dans les étapes suivantes.
- Ajouter une activité *DestinataireImpliciteActivity* avec son interface contenant un textview affichant ***Activité Destinataire implicite***.

- Modifie le fichier AndroidManifest pour avoir le code suivant par rapport à l'activité *DestinataireImpliciteActivity*

```
<activity android:name=".DestinataireImpliciteActivity"
android:label="@string/title_activity_destinataire_implicit">
    <intent-filter>
        <category android:name="android.intent.category.DEFAULT" />
        <action android:name="ca.qc.cgodin.laboratoire2.ACTION1" />
        <category android:name="ca.qc.cgodin.laboratoire2.CATEGORIE1" />
    </intent-filter>
</activity>
```

  Prenez note que la ligne suivante

```
<category android:name="android.intent.category.DEFAULT" />
```

  doit être présente dans tous les composants prêts à répondre à des intentions implicites. Toute catégorie d'un objet *Intent* doit être listée dans un *intent-filter.*

- Ajoutez la déclaration d'ACTION2 dans le fichier *AndroidManifest* pour avoir ce qui suit:

```
<activity android:name=".DestinataireImpliciteActivity"
android:label="@string/title_activity_destinataire_implicit">
    <intent-filter>
        <category android:name="android.intent.category.DEFAULT" />
        <action android:name="ca.qc.cgodin.laboratoire2.ACTION1" />
        <category android:name="ca.qc.cgodin.laboratoire2.CATEGORIE1" />
        <category android:name="android.intent.category.DEFAULT" />
        <action android:name="ca.qc.cgodin.laboratoire2.ACTION2" />
    </intent-filter>
</activity>
```

- Comme pour le bouton *ACTION1*, ajouter la méthode suivante à *MainActivity,* pour répondre au click du bouton *ACTION2.*

```
fun onClickButtonAction2(view: View) {
    val intent = Intent("ca.qc.cgodin.laboratoire2.ACTION2")
    startActivity(intent)
}
```

- Pour savoir quelle action a été choisie, ajoutez le code suivant à *DestinataireImpliciteActivity* dans la méthode *onCreate()*

```
val action = intent().action;
Toast.makeText(this, " Vous avez choisi l'action: " + action, Toast.LENGTH_LONG).show();
```

- Exécuter à nouveau, puis cliquer sur le bouton *ACTION1* puis sur le bouton *ACTION2.* Observez l'affichage des toasts.

- Programmer le bouton ACTION_VIEW en ajoutant le code suivant à *MainActivity*

```
fun onClickButtonActionView(view: View) {
    val intent = Intent(Intent.ACTION_VIEW)
    intent.data = Uri.parse("http://www.cgodin.qc.ca")
    startActivity(intent)
}
```

- Exécuter puis cliquer sur le bouton ACTION_VIEW. La page d'accueil du site du collège s'affiche.

- Comme pour le bouton ACTION1 et ACTION2, ajoutez la déclaration d'ACTION_VIEW dans le fichier AndroidManifest.xml comme suit.

```
<activity
    android:name=".DestinataireImpliciteActivity"
    android:label="@string/title_activity_destinataire_implicit">
    <intent-filter>
        <category android:name="android.intent.category.DEFAULT" />
        <action android:name="ca.qc.cgodin.laboratoire2.ACTION1" />
        <category android:name="ca.qc.cgodin.laboratoire2.CATEGORIE1" />
        <category android:name="android.intent.category.DEFAULT" />
        <action android:name="ca.qc.cgodin.laboratoire2.ACTION2" />
    </intent-filter>
    <intent-filter>
        <category android:name="android.intent.category.DEFAULT" />
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="http"/>
    </intent-filter>
</activity>
```

- Exécuter. Cette fois-ci, le système vous offre un choix à l'utilisateur car il a rencontré deux composants applicatifs susceptibles de répondre à l'action *ACTION_VIEW*.

  Un **filtre d'intention** (Intent-Filter) sert à préciser les modalités de dialogues entre une intention (un besoin) et votre application (une offre de service).