

Programmation mobile

Cours 2 : Intent

Julien Grange <julien.grange@lacl.fr>

Mardi 24 septembre 2024

Rappel : lancer une `Activity` sans attendre de résultat

Dans l'`Activity` appelante :

```
int value1 = ...;
String value2 = ...;

// Intent explicite
Intent intent = new Intent(context, NewActivity.class);
intent.putExtra("Key1", value1);
intent.putExtra("Key2", value2);

startActivity(intent);
```

Dans `NewActivity` :

```
Intent intent = getIntent();

int myInt = intent.getIntExtra("Key1", 42);
String myString = intent.getStringExtra("Key2");
```

Quatre façons de lancer une `Activity`

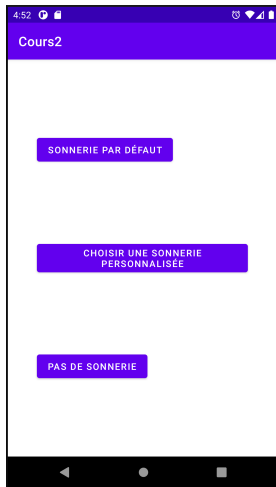
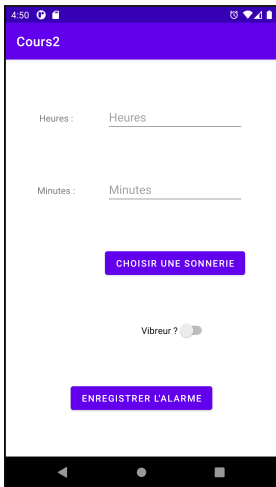
Deux types d'`Intent` :

- explicite (en nommant l'`Activity`)
- implicite (en spécifiant le type d'action)

Deux manières de lancer une `Activity` :

- sans attendre de résultat : `startActivity()`
- dans l'attente d'un résultat : `registerForActivityResult()`

Application du jour : Alarme



Trois appels à une autre `Activity` :

- 1 “Choisir une sonnerie” : explicite, avec résultat
- 2 “Choisir une sonnerie personnalisée” : implicite, avec résultat
- 3 “Enregistrer l’alarme” : implicite, sans résultat

On a déjà vu les appels explicites sans résultat.

MainActivity layout

The screenshot shows the Android Studio IDE with the following components:

- Code Editor:** Contains XML for `activity_main.xml`. It defines two `EditText` fields: `hours` (with hint "Heures") and `minutes` (with hint "Minutes"). Both are constrained to a parent layout with various alignment and spacing constraints.
- Layout Editor:** A visual diagram showing the layout structure. It includes a `ConstraintLayout` container with three child views: `hours`, `minutes`, and `ENREGISTRER L'ALARME`. The `hours` and `minutes` fields are positioned at the top, and the button is at the bottom.
- Attributes Panel:** Shows the attributes for the selected `hours` `EditText` widget, including `layout_width`, `layout_height`, `wrap_content`, `android:ems`, `android:hint`, `android:inputType`, `android:id`, and `android:inputType`.
- Constraints Panel:** Lists the constraints for the selected widget, such as `layout_constraintBottom_toBottomOf`, `layout_constraintEnd_toEndOf`, `layout_constraintHorizontal_bias`, `layout_constraintStart_toStartOf`, and `layout_constraintTop_toTopOf`.
- Bottom Bar:** Shows the status bar with "Launch succeeded (9 minutes ago)" and various tool icons.

MainActivity code

The screenshot displays the Android Studio IDE with the MainActivity.java file open. The code defines a MainActivity class that extends AppCompatActivity. It includes fields for EditText (editHours, editMinutes), Button (buttonRingtone, buttonAdd), and Switch (switchVibreur). The class also defines several constants for ringtone types and implements the onCreate method to initialize the UI and register for an activity result.

```
1 package con.example.cours2;
2
3 import ...
4
24 public class MainActivity extends AppCompatActivity {
25
26
27     private EditText editHours, editMinutes;
28     private Button buttonRingtone, buttonAdd;
29     private Switch switchVibreur;
30
31     public static final String
32         NOTIFY = "con.example.cours2.NOTIFY",
33         RINGTONE = "con.example.cours2.RINGTONE",
34         DEFAULT = "con.example.cours2.DEFAULT",
35         CHOSEN = "con.example.cours2.CHOSEN",
36         NO = "con.example.cours2.NO";
37
38     private String typeRingtone=NOTIFY;
39     private Uri ringtone;
40
41     /*...*/
42     private ActivityResultCallback<Intent> callback = new ActivityResultCallback<Intent> (...);
43
44     private ActivityResultContract<Void, Intent> contract = new ActivityResultContract<Void, Intent> (...);
45
46     // Lancer pour debuter l'activite Ringtone
47     ActivityResultLauncher<Void> selectRingtone = registerForActivityResult(contract, callback);
48
49     @Override
50     protected void onCreate(Bundle savedInstanceState) {...}
51
52     //on attrape les vies declares dans le XML
53     private void catchViews(){...}
54
55     private void ringtoneIssue() {...}
56     private void formatIssue() {...}
57
58 }
```

The UI preview on the right shows a smartphone emulator with the following elements:

- Time: 12:20
- Title: Cours2
- Hours: Heures (input field)
- Minutes: Minutes (input field)
- Button: CHOISIR UNE SONNERIE
- Switch: Vibreur ?
- Button: ENREGISTRER L'ALARME

registerForActivityResult()

The screenshot displays the Android Studio IDE with the following components:

- Code Editor:** Shows the implementation of `registerForActivityResult` in `MainActivity.java`. The code includes a `callback` object and a `contract` object, both implementing `ActivityResultCallback` and `ActivityResultContract` respectively. The `onActivityResult` method handles the result, and the `registerForActivityResult` call is used to register the contract and callback.
- UI Preview:** A virtual device showing the app's interface. The screen has a purple header with the title "Cours2". Below the header, there are two input fields labeled "Heures:" and "Minutes:". A purple button labeled "CHOISIR UNE SONNERIE" is positioned below the input fields. At the bottom of the screen, there is a toggle switch labeled "Vibreur ?" and another purple button labeled "ENREGISTRER L'ALARME".
- IDE Interface:** The top menu bar includes File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, and Help. The bottom status bar shows "Launch succeeded (8 minutes ago)", "7:48", "LF", "UTF-8", and "4 spaces".

registerForActivityResult()

Paramétré par deux classes : `I` (input) et `O` (output)

```
public ActivityResultLauncher<I> registerForActivityResult  
    (ActivityResultContract<I,O> contract,  
     ActivityResultCallback<O> callback)
```

registerForActivityResult()

Paramétré par deux classes : `I` (input) et `O` (output)

```
public ActivityResultLauncher<I> registerForActivityResult  
    (ActivityResultContract<I,O> contract,  
     ActivityResultCallback<O> callback)
```

- callback : implémente l'interface `ActivityResultCallback<O>`

```
public void onActivityResult(O result)
```

registerForActivityResult()

Paramétré par deux classes : `I` (input) et `O` (output)

```
public ActivityResultLauncher<I> registerForActivityResult  
(ActivityResultContract<I,O> contract,  
 ActivityResultCallback<O> callback)
```

- callback : implémente l'interface `ActivityResultCallback<O>`

```
public void onActivityResult(O result)
```

- contract : implémente l'interface `ActivityResultContract<I,O>`

```
public Intent createIntent(Context context, I input)  
public O parseResult(int resultCode, Intent intent)
```

registerForActivityResult()

Paramétré par deux classes : `I` (input) et `O` (output)

```
public ActivityResultLauncher<I> registerForActivityResult  
(ActivityResultContract<I,O> contract,  
 ActivityResultCallback<O> callback)
```

- callback : implémente l'interface `ActivityResultCallback<O>`

```
public void onActivityResult(O result)
```

- contract : implémente l'interface `ActivityResultContract<I,O>`

```
public Intent createIntent(Context context, I input)  
public O parseResult(int resultCode, Intent intent)
```

- l'appel peut ensuite être fait via la méthode `launch(I input)` du résultat de `registerForActivityResult()`

registerForActivityResult()

Paramétré par deux classes : `I` (input) et `O` (output)

```
public ActivityResultLauncher<I> registerForActivityResult  
(ActivityResultContract<I,O> contract,  
 ActivityResultCallback<O> callback)
```

- `callback` : implémente l'interface `ActivityResultCallback<O>`

```
public void onActivityResult(O result)
```

- `contract` : implémente l'interface `ActivityResultContract<I,O>`

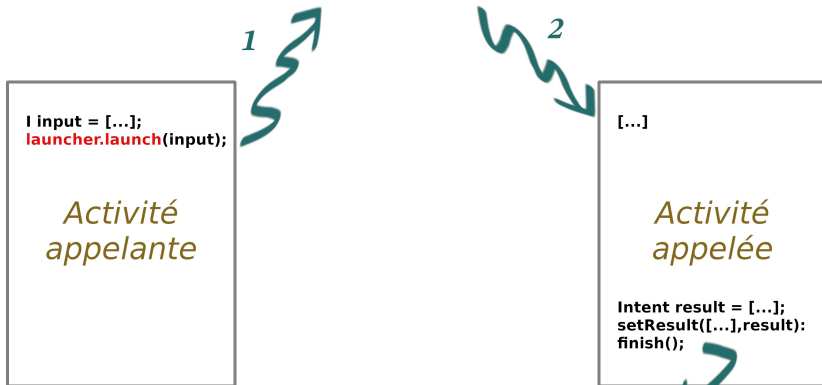
```
public Intent createIntent(Context context, I input)  
public O parseResult(int resultCode, Intent intent)
```

- l'appel peut ensuite être fait via la méthode `launch(I input)` du résultat de `registerForActivityResult()`

Le callback doit être défini dès la création de l'`Activity` appelante.

registerForActivityResult() : derrière la scène

```
Intent intent = contract.createIntent(input);
```



```
O output = contract.parseResult(result);
```

```
callback.onActivityResult(output);
```

registerForActivityResult()

Ici, **I** = **Void** (aucun input) et **O** = **Intent**.

The screenshot displays the Android Studio IDE with the following components:

- Code Editor:** Shows the implementation of `registerForActivityResult` in `MainActivity.java`. The code defines a `callback` and a `contract` to register for an `ActivityResultContract` that returns an `Intent`. The `onActivityResult` method handles the result, specifically checking for a ringtone selection.
- UI Preview:** A virtual smartphone displays the app's interface, titled "Cours2". It features two input fields for "Heures" and "Minutes", a "CHOISIR UNE SONNERIE" button, and a "Vibreur ?" toggle switch. At the bottom, there is an "ENREGISTRER L'ALARME" button.
- IDE Interface:** The top toolbar shows the "Run" button (a green play icon) and other development tools. The bottom status bar indicates "Launch succeeded (8 minutes ago)".

registerForActivityResult()

```
74 //à lancer pour débiter l'activité Ringtone
75 ActivityResultLauncher<Void> selectRingtone = registerForActivityResult(contract, callback);
76
77
78 @Override
79 protected void onCreate(Bundle savedInstanceState) {
80     super.onCreate(savedInstanceState);
81     setContentView(R.layout.activity_main);
82     catchViews();
83
84     buttonRingtone.setOnClickListener(new View.OnClickListener() {
85         @Override
86         public void onClick(View view) { selectRingtone.launch(input: null); }
87     });
88
89     buttonAdd.setOnClickListener(new View.OnClickListener() { .. });
90
129
130
131
132 //on attrape les views déclarées dans le XML
133 private void catchViews(){
134     editHours = findViewById(R.id.hours);
135     editMinutes = findViewById(R.id.minutes);
136     buttonRingtone = findViewById(R.id.ringtone);
137     switchVibreur = findViewById(R.id.vibreur);
138     buttonAdd= findViewById(R.id.add);
139 }
140
```

The screenshot shows the Android Studio interface. On the left, the Java code for MainActivity.java is displayed, showing the registration of an ActivityResultLauncher and the implementation of onCreate and catchViews methods. On the right, a preview of the app's UI is shown on a smartphone. The UI features a purple header with the text 'Cours2', two input fields for 'Heures' and 'Minutes', a button labeled 'CHOISIR UNE SONNERIE', a 'Vibreur ?' toggle switch, and a button labeled 'ENREGISTRER L'ALARME'. The bottom status bar shows the time as 7:48, location as LF, and other system icons.

Ici, `I` = `String` (MIME type) et `O` = `Uri` (fichier audio).

The screenshot displays the Android Studio IDE with the following components:

- Code Editor:** Shows the `Ringtone.java` file. The code defines a class `Ringtone` extending `AppCompatActivity`. It includes three buttons: `buttonDefault`, `buttonChoose`, and `buttonNo`. The `onActivityResult` method is overridden to handle the selection of a ringtone file. The `onCreate` method is protected and currently empty. The `sendRingtone` method is also defined but empty.
- Preview:** A virtual device (Pixel XL API 30) is shown on the right, displaying the app's UI. The screen has a purple header with the text "Cours2". Below the header are three buttons: "SONNERIE PAR DÉFAUT", "CHOISIR UNE SONNERIE PERSONNALISÉE", and "PAS DE SONNERIE".
- IDE Interface:** The top menu bar includes File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, and Help. The bottom status bar shows "Launch succeeded (11 minutes ago)", "Event Log", "Layout Inspector", and "35:56 LF UTF-8 4 spaces".

The screenshot displays the Android Studio IDE with the following components:

- Code Editor:** Shows the `onCreate` method of `MainActivity` in `Ringtone.java`. The code includes:
 - Initialization of buttons: `buttonDefault`, `buttonChoose`, and `buttonNo`.
 - Setting of `buttonDefault` and `buttonChoose` click listeners. The `onClick` methods call `sendRingtone()` and pass specific ringtone names (`DEFAULT`, `CHOSEN`, `NO`).
 - A `sendRingtone()` private method that sets the result and finishes the activity.
- Preview:** A virtual smartphone displays the app's UI with a purple header labeled "Cours2" and three buttons: "SONNERIE PAR DÉFAUT", "CHOISIR UNE SONNERIE PERSONNALISÉE", and "PAS DE SONNERIE".
- Bottom Bar:** Shows the status "Launch succeeded (12 minutes ago)" and system information: "35:56 LF UTF-8 4 spaces".

Pour passer un résultat à l'`Activity` appelante :

```
Intent result = new Intent();
result.putExtra(...);
result.setData(data);

setResult(code, result);
finish();
```

où code est `Activity.RESULT_OK` ou `Activity.RESULT_CANCELED`.

MainActivity Enregistrement de l'alarme

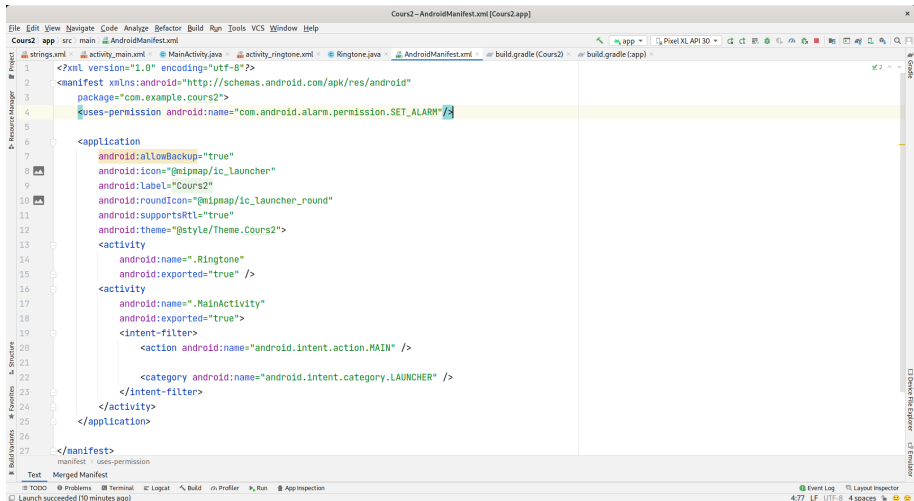
The image shows the Android Studio IDE with the MainActivity.java file open. The code implements an alarm registration feature. It includes a button click listener that reads hours and minutes from text inputs, validates the time (between 00:00 and 24:00), and sets an alarm using AlarmClock. The alarm can be set to ringtone, chosen, or silent. A 'CHOISIR UNE SONNERIE' button and a 'Vibreur ?' toggle are also present. The 'ENREGISTRER L'ALARME' button triggers the alarm registration logic.

```
buttonAdd.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        int hours, minutes;  
        try {  
            hours = Integer.valueOf(String.valueOf(editHours.getText()));  
            minutes = Integer.valueOf(String.valueOf(editMinutes.getText()));  
  
            if(0 <= hours && 0 <= minutes && hours < 24 && minutes < 60){  
                // l'heure est valide : on lance l'alarme  
                Intent setAlarm = new Intent(AlarmClock.ACTION_SET_ALARM);  
                setAlarm.putExtra(AlarmClock.EXTRA_HOUR, Integer.valueOf(hours));  
                setAlarm.putExtra(AlarmClock.EXTRA_MINUTES, Integer.valueOf(minutes));  
                setAlarm.putExtra(AlarmClock.EXTRA_VIBRATE, switchVibreur.isChecked());  
                switch (typeRingtone){  
                    case DEFAULT:  
                        //si pas de EXTRA_RINGTONE, la sonnerie par defaut est utilisée  
                        startActivity(setAlarm);  
                        break;  
                    case CHOSEN:  
                        setAlarm.putExtra(AlarmClock.EXTRA_RINGTONE, ringtone);  
                        startActivity(setAlarm);  
                        break;  
                    case NO:  
                        setAlarm.putExtra(AlarmClock.EXTRA_RINGTONE, AlarmClock.VALUE_RINGTONE_SILENT);  
                        startActivity(setAlarm);  
                        break;  
                    default:  
                        ringtoneIssue();  
                }  
            } else {  
                formatIssue();  
            }  
        } catch (NumberFormatException e) {  
            formatIssue();  
        }  
    }  
});
```

The mobile emulator displays the app interface with the following elements:

- Time: 12:17
- Title: Cours2
- Hours: _____
- Minutes: _____
- CHOISIR UNE SONNERIE button
- Vibreur ?
- ENREGISTRER L'ALARME button

On demande la permission `“com.android.alarm.permission.SET_ALARM”`.



```
Cours2 - AndroidManifest.xml [Cours2.app]
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Cours2 app src: main AndroidManifest.xml
strings.xml activity_main.xml MainActivity.java activity_ringtones.xml Ringtone.java AndroidManifest.xml build.gradle (Cours2) build.gradle (app)
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3 package="com.example.cours2">
4 <uses-permission android:name="com.android.alarm.permission.SET_ALARM" />
5
6 <application
7     android:allowBackup="true"
8     android:icon="@mipmap/ic_launcher"
9     android:label="Cours2"
10    android:roundIcon="@mipmap/ic_launcher_round"
11    android:supportRtl="true"
12    android:theme="@style/Theme.Cours2">
13    <activity
14        android:name=".Ringtone"
15        android:exported="true" />
16    <activity
17        android:name=".MainActivity"
18        android:exported="true">
19        <intent-filter>
20            <action android:name="android.intent.action.MAIN" />
21
22            <category android:name="android.intent.category.LAUNCHER" />
23        </intent-filter>
24    </activity>
25 </application>
26
27 </manifest>
manifest uses-permission
Text Merged Manifest
Event Log Layout Inspector
4:77 LF UTF-8 4 spaces
```

Signaler qu'on sait traiter un `Intent` implicite

Pour déclarer qu'on sait envoyer des images par mail, ajouter dans le Android Manifest :

```
<activity android:name="MyMailer">
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="image/*"/>
  </intent-filter>
</activity>
```