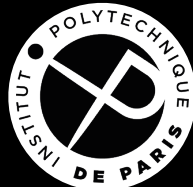


# Développement d'applications mobiles avec Android : **Activités & Intents**

James EAGAN

[james.eagan@telecom-paristech.fr](mailto:james.eagan@telecom-paristech.fr)



Mise à jour : octobre 2019.

1

## Agenda

- Introduction
- Cycle de vie d'une appli Android
- Gestion d'état entre instances
- Communication entre applis

2

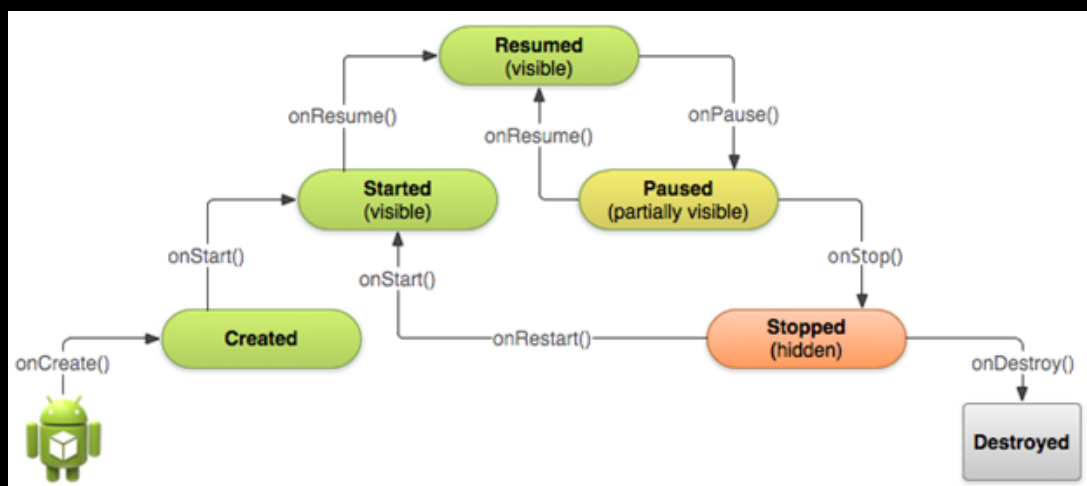
# Activity

- Une activité est l'unité d'une application
- On peut le considérer un peu comme une fenêtre

3

3

# Cycle de vie d'une Activity

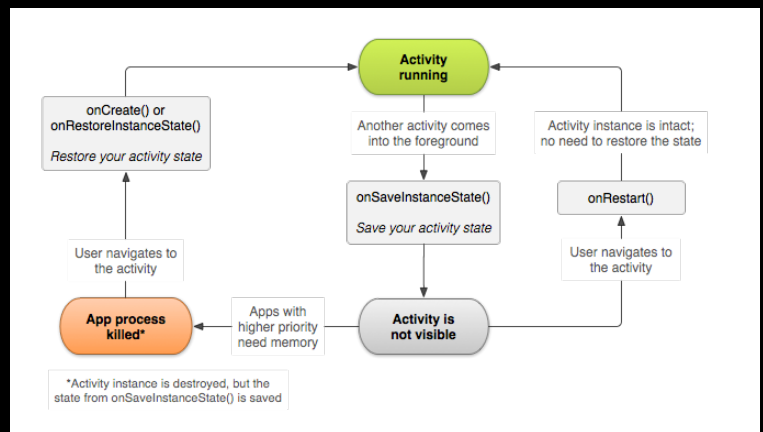


4

4

# Persistence d'état entre vies

- Quand une `Activity` est tuée, son état peut être stocké dans un `Bundle`.
- Un `Bundle` est un paquet qui peut stocker des objets `Parcelable` (un peu comme `Serializable`).
- L'implémentation par défaut appelle `onSaveInstanceState()` sur les éléments de l'IU.



5

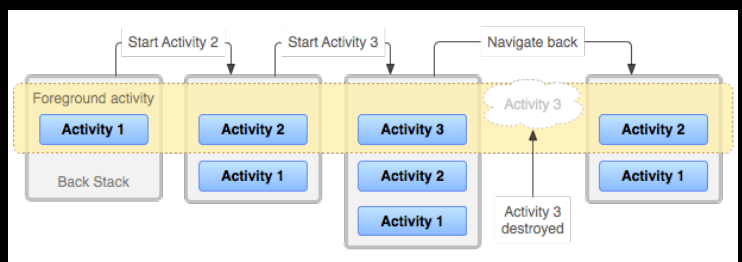
# Transfert d'état entre activités

- Généralement, on sauvegarde l'état à chaque `onPause()`, car l'activité pourrait être tuée à n'importe quel moment après. Si l'activité en lance une autre, Android garantit que `onPause()` sera appelée avant les méthodes de la nouvelle `Activity` : `onCreate()`, `onStart()`, et `onResume()`.

6

# Gestion du bouton ↩

- Android maintient une pile de toute tâche (avec une interface cohérente) pour le bouton retour-en-arrière.
- Parfois, il est souhaitable de modifier ce comportement :
  - Activités instanciées plusieurs fois ou pas du tout
  - Lancer une nouvelle activité implique (ou pas) lancer une nouvelle application



7

7

# Intent : communication entre composants

- Une *Intent* exprime une action et ses paramètres :
  - Une *Intent explicite* vise un composant connu avec son package et nom de classe.
  - Une *Intent implicite* décrit une opération à faire, mais pas qui doit la faire.

8

8

# Quand utiliser les deux

- À l'intérieur d'une appli, on utilise généralement des **Intents** explicites pour naviguer entre **Activities**.
- Pour scanner un code QR, on utilise généralement des **Intents** implicites pour trouver un service installé dans le système.
- Pour partager quelque chose, on utilise un chooser qui liste les composants capable de le partager (e.g. mail, sms, twitter, ...)
- Il pourrait y en avoir plusieurs : pour trouver un chooser, on utilise une **Intent** implicite en donnant l'**Intent** de partage comme paramètre.

9

9

# Intent

Une Intent a plusieurs attributs :

**Nom** l'**Activity** à démarrer—utilisé pour des **Intents** explicites

**Data** un URI (si besoin) et un type MIME (e.g. text/plain)

**Catégorie** la catégorie des genres de composants qui pourraient gérer cette action (e.g. CATEGORY\_LAUNCHER pour des activités qui peuvent être lancées depuis l'écran d'accueil).

**Extras** Paramètres supplémentaires (clé, valeur)

**Drapeaux** infos supplémentaires (e.g. s'il faut lancer une nouvelle tâche).

10

# Exemple d'une Intent explicite

- Appel :

```
final Intent editIntent = new Intent(this, RecipeEditor.class);
editIntent.putExtra(EXTRA_RECIPE_ID, recipe.getId());
startActivity(editIntent);
```

- Définition :

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    final int recipe = getIntent().getIntExtra(EXTRA_RECIPE_ID, 0);
    // ...
}
```

11

11

# Exemple d'une Intent implicite avec Chooser

```
// Créer une Intent implicite pour partager du texte
final Intent intent = (new Intent(Intent.ACTION_SEND))
    .setType("text/plain")
    .putExtra(Intent.EXTRA_TEXT, recipeText.toString());

// Lancer un chooser pour laisser choisir l'application à utiliser
// pour le partage. Intent.createChooser() est une méthode statique
// qui construit une Intent implicite de type ACTION_CHOOSER.
final String title = res.getText(R.string.recipe_share);
startActivity(Intent.createChooser(intent, title));
```

12

12

# Filtres d'Intent

- Pour recevoir une Intent, elle doit être déclarée dans le `AndroidManifest.xml`

```
<activity
  android:name=".RecipeManagerActivity"
  android:label="@string/app_name">
  <intent-filter>
    <action android:name="android.intent.action.MAIN"/>
    <category android:name="android.intent.category.LAUNCHER"/>
  </intent-filter>
</activity>
```

[ <https://developer.android.com/guide/components/intents-filters.html> ]

13

13

# Un exemple plus complexe

```
<intent-filter>
  <action android:name="android.intent.action.VIEW"/>
  <category android:name="android.intent.category.DEFAULT"/>
  <data android:mimeType="text/xml"/>
  <data android:mimeType="application/xml"/>
  <data android:mimeType="application/zip"/>
  <data android:mimeType="application/x-compressed"/>
  <data android:mimeType="application/x-zip-compressed"/>
  <data android:mimeType="application/x-zip"/>
  <data android:mimeType="application/octet-stream"/>
  <data android:pathPattern=".*\\.recipe"/>
  <data android:pathPattern=".*\\.zip"/>
</intent-filter>
```

14

14

# Recevoir le résultat d'une Activity

- Parfois, on veut lancer une Activity et récupérer le résultat (e.g. Scanner un flashcode)
- Pour lancer l'Activity : `startActivityForResult()` avec un code de requête
- Dans l'Activity lancé : `setResult()`
- On reçoit le résultat dans la méthode `onActivityResult()` (avec le code de requête utilisé dans `startActivityForResult()`)

15

15

# Exemple : demander un 06

```
static final int PICK_CONTACT_REQUEST = 1; // The request code

private void pickContact() {
    Intent pickContactIntent = new Intent(Intent.ACTION_PICK, Uri.parse("content://contacts"));
    pickContactIntent.setType(Phone.CONTENT_TYPE); // Show only contacts w/ phone numbers
    startActivityForResult(pickContactIntent, PICK_CONTACT_REQUEST);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // Check which request we're responding to
    if (requestCode == PICK_CONTACT_REQUEST) {
        // Make sure the request was successful
        if (resultCode == RESULT_OK) {
            // The user picked a contact.
            // The Intent's data Uri identifies which contact was selected.

            // Do something with the contact here (bigger example below)
        }
    }
}
```

16

16



```

protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // Check which request we're responding to
    if (requestCode == PICK_CONTACT_REQUEST) {
        // Make sure the request was successful
        if (resultCode == RESULT_OK) {
            // Get the URI that points to the selected contact
            Uri contactUri = data.getData();
            // We only need the NUMBER column, because there will be only one result
            String[] projection = {Phone.NUMBER};

            // Perform the query on the contact to get the NUMBER column
            // We don't need a selection or sort order for only one result
            // CAUTION: The query() method should be called from a separate thread to
            // avoid blocking your app's UI thread.
            // Consider using CursorLoader to perform the query.
            Cursor cursor = getContentResolver()
                .query(contactUri, projection, null, null, null);
            cursor.moveToFirst();

            // Retrieve the phone number from the NUMBER column
            int column = cursor.getColumnIndex(Phone.NUMBER);
            String number = cursor.getString(column);

            // Do something with the phone number...
        }
    }
}

```

17

17

# Gestion de Threads

18

18

# Rappel : UI Thread

- Toute interaction se passe sur le fil d'exécution principale (UI Thread)
- Les méthodes de rappel (listeners) doivent terminer rapidement
- Que fait-on si on a une tâche longue ?

Activity.**runOnUiThread**(runnable)

**AsyncTask**

19

19

## Activity.runOnUiThread(runnable)

```
public void triggerDownload(View sender) {  
    downloadData(); // May take a long time  
}
```

20

20

## Activity.runOnUiThread(runnable)

```
public void triggerDownload(View sender) {
    new Thread(new Runnable() {
        public void run() {
            downloadData();
        }
    }).start();
}

private void downloadData() {
    /* Download data here... */ // This might take a while
    registerNewData(); // When done, record new data
}
```

21

21

## Activity.runOnUiThread(runnable)

```
public void triggerDownload(View sender) {
    // new Thread(new Runnable() {
    //     public void run() {
    //         downloadData();
    //     }
    // }).start();
}

private void downloadData() {
    /* Download data here... */ // This might take a while
    registerNewData(); // But if this updates the UI,
                       // needs to run on main thread!
}
```

22

22

## Activity.runOnUiThread(runnable)

```
public void triggerDownload(View sender) {
    new Thread( () → downloadData() ).start();
}

private void downloadData() {
    /* Download data here... */ // This might take a while
    registerNewData(); // But if this updates the UI,
                       // needs to run on main thread!
}

private void registerNewData() {
    // Must be run on UI Thread; acts on new data
}
```

23

23

## Activity.runOnUiThread(runnable)

```
public void triggerDownload(View sender) {
    new Thread( () → downloadData() ).start();
}

private void downloadData() {
    /* Download data here... */ // This might take a while
    runOnUiThread( () → registerNewData() );
}

private void registerNewData() {
    // Must be run on UI Thread; acts on new data
}
```

24

24



# AsyncTask

```
private class DownloadTask extends AsyncTask<URL, Void, JSONObject> {
    @Override
    protected JSONObject doInBackground(URL... urls) {

    }


    @Override
    protected void onPostExecute(JSONObject result) {

    }
}
```

27

# AsyncTask

```
private class DownloadTask extends AsyncTask<URL, Integer, JSONObject> {
    protected JSONObject doInBackground(URL... urls) {

        return downloadData();
    }

    protected void onPostExecute(JSONObject result) {
        registerNewData();
    }

    protected void onProgressUpdate(Integer... values) {
        // runs on UI thread ; update Integer progress
    }
}
```

28

# JSONObject

29

29

## JSON

```
{ "menu": {  
  "id": 34714636,  
  "label": "File",  
  "popup": {  
    "menuitem": [  
      { "value": "New", "onclick": "CreateNewDoc()" },  
      { "value": "Open", "onclick": "OpenDoc()" },  
      { "value": "Close", "onclick": "CloseDoc()" }  
    ]  
  }  
}
```

- Strings
- Numbers
- Dictionaries/Objects — key: value
- Arrays

30

30

# JSONObject

- Classe Java pour gérer le JSON
- `JSONObject(jsonString)` lire à partir d'un String
- `jsonObject.toString()` générer du JSON
- `jsonObject.put(name, value)` muter une valeur
- `jsonObject.getType(name)` obtenir une valeur (throws)
- `jsonObject.optType(name)` obtenir une valeur (peut être "null")
  - also `jsonObject.optType(name, default)`
  - and `jsonObject.get()` obtenir un Object Java
- Type parmi : `boolean`, `double`, `integer`, `long`, `String`, `JSONArray`, `JSONObject`

31

31

# JSONArray

- Classe Java pour gérer un tableau JSON
- `jsonArray.toString()` générer du JSON
- `jsonArray.put(value)` rajouter une valeur
- `jsonArray.put(i, value)` muter une valeur
- `jsonArray.getType(i)` obtenir une valeur (throws)
- `jsonArray.optType(i)` obtenir une valeur (peut être "null")
  - also `jsonArray.optType(i, default)`
  - and `jsonArray.get(i)` obtenir un Object Java
- Type parmi : `boolean`, `double`, `integer`, `long`, `String`, `JSONArray`, `JSONObject`

32

32



[developer.android.com](http://developer.android.com)



[ Transparents adaptés de ceux de Samuel Tardieu ]