# Communication sous android

# Plan

# I. Introduction

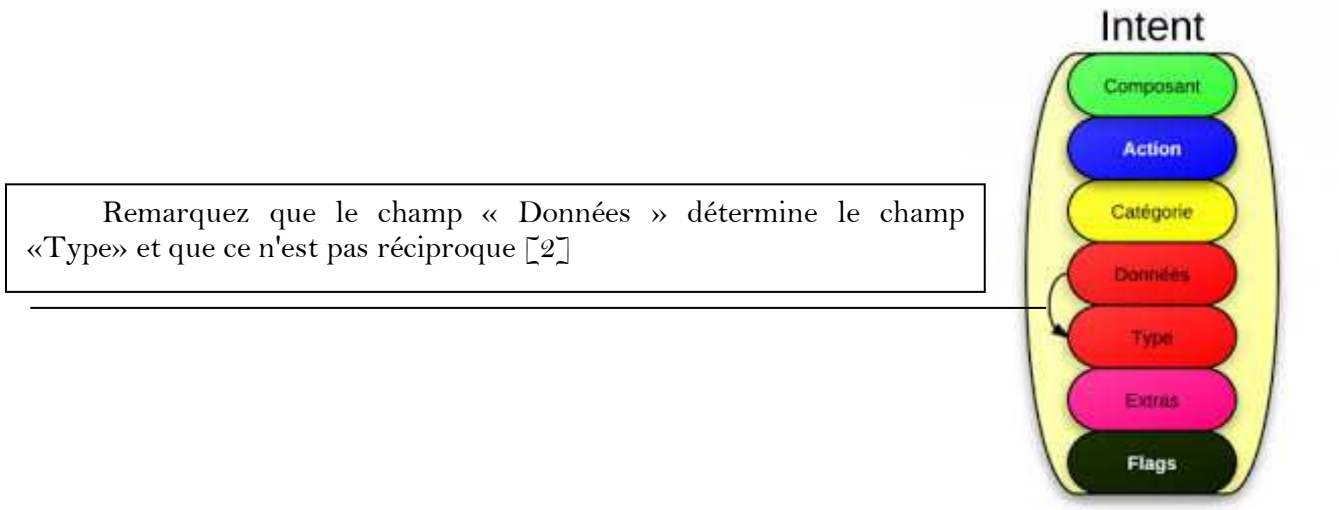
La communication interne d'Android est basée sur l'envoi et la réception de messages exprimant une opération à effectuer. Les messages peuvent être émis à destination d'un autre composant de la même application (une activité, un service…) ou vers n'importe quelle autre application. La classe android.content.Intent modélise un tel message [1].

Dans android, chaque application est censée vivre dans un compartiment cloisonné pour ne pas déranger le système quand elle s'exécute et surtout quand elle plante. Le mécanisme d'échange de message rend android un vrai puzzle dont chaque pièce apporte une fonctionnalité qui pourrait fournir son aide à une autre pièce, ou au contraire qui aurait besoin de l'aide d'une autre pièce [2].

Les agents qui sont chargés du mécanisme d'échange de messages s'appellent les intents. Par exemple, si l'utilisateur clique sur un numéro de téléphone dans votre application, peut-être souhaiteriez-vous que le téléphone appelle le numéro demandé. Avec un intent, vous allez dire à tout le système que vous avez un numéro qu'il faut appeler, et c'est le système qui fera en sorte de trouver les applications qui peuvent le prendre en charge. Ce mécanisme est tellement important qu'Android lui-même l'utilise massivement en interne [2].

# II. Intent

Un intent est en fait un objet qui contient plusieurs champs, représentés à la figure suivante [2] :




Remarquez que le champ « Données » détermine le champ «Type» et que ce n'est pas réciproque [2]

La façon dont ces champs sont renseignés détermine la nature ainsi que les objectifs de l'intent. Ainsi, pour qu'un intent soit dit « **explicite** », il suffit que son champ **composant** soit renseigné. Ce champ permet de définir le destinataire de l'intent, celui qui devra le gérer. Ce champ est constitué de deux informations : le package où se situe le composant, ainsi que le nom du composant. Ainsi, quand l'intent sera exécuté, Android pourra retrouver le composant de destination de manière précise [2].

À l'opposé des intents explicites se trouvent les intents « **implicites** ». Dans ce cas de figure, on ne connaît pas de manière précise le destinataire de l'intent [2].

## II.1. Les intents explicites

Pour créer un intent explicite il suffit de donner un Context qui appartient au package où se trouve la classe de destination [2]:

```
Intent intent = new Intent(Context context, Class<?> cls);
```

Par exemple, si la classe de destination appartient au package du Context actuel :

```
Intent intent = new Intent(Activite_de_depart.this, Activite_de_destination.class);
```

## II.2. Les intents implicites

Pour les intents implicites, on fera en sorte d'envoyer une requête à un destinataire, sans savoir qui est et c'est à Android de déterminer et de lancer l'application qui exécute la requête (Par exemple : Envoyer un SMS sans préciser l'application qui va être utlisée). Les applications destinataires sont : soit fournies par Android, soit par d'autres applications téléchargées sur le Play Store par exemple [2].

Pour que Android puisse déterminer qui est capable de réceptionner un intent implicite. Il faut au moins fournir deux informations essentielles [2]:

- **Une action** : ce qu'on désire que le destinataire fasse.
- **Un ensemble de données** : sur quelles données le destinataire doit effectuer son action.

Il existe aussi d'autres informations, pas forcément obligatoires, mais qui ont aussi leur utilité propre le moment venu [2]:

- **La catégorie** : permet d'apporter des informations supplémentaires sur l'action à exécuter et le type de composant qui devra gérer l'intent.
- **Le type** : pour indiquer quel est le type des données incluses. Normalement, ce type est contenu dans les données, mais en précisant cet attribut vous pouvez désactiver cette vérification automatique et imposer un type particulier.
- **Les extras** : pour ajouter du contenu à vos intents afin de les faire circuler entre les composants.
- **Les flags** : permettent de modifier le comportement de l'intent.

### II.2.1. L'action

Une action est une constante qui se trouve dans la classe Intent et qui commence toujours par «ACTION_» suivi d'un verbe (en anglais) de façon à bien faire comprendre qu'il s'agit d'une action. Si on veut voir quelque chose, on va utiliser l'action ACTION_VIEW [2]. Par exemple, si on utilise ACTION_VIEW sur un numéro de téléphone, alors le numéro de téléphone s'affichera dans le composeur de numéros de téléphone [2].

Voici quelques actions natives parmi les plus utilisées [2]:

| Intitulé | Action | Entrée attendue |
|---|---|---|
| ACTION_MAIN | Pour indiquer qu'il s'agit du point d'entrée dans l'application | - |
| ACTION_DIAL | Pour ouvrir le composeur de numéros téléphoniques | Un numéro de téléphone |
| ACTION_SEARCH | Effectuer une recherche | Le texte à rechercher |
| ACTION_SENDTO | Envoyer un message à quelqu'un | Le numéro de la personne à qui envoyer le message |
| ACTION_VIEW | Visionner une donnée | Plusieurs utilisations possibles : Une adresse e-mail sera visionnée dans l'application pour les e-mails, un numéro de téléphone dans le composeur, une adresse internet dans le navigateur, etc. |
| ACTION_WEB_SEARCH | Effectuer une recherche sur internet | S'il s'agit d'un texte qui commence par « http », le site s'affichera directement, sinon c'est une recherche dans Google qui se fera |

### II.2.2. Les données

Généralement les données sont envoyées sous forme d'URI (Uniform Resource Identifier).

Les URI se comportent d'une manière un peu similaire. La syntaxe d'un URI peut être analysée de la manière suivante (les parties entre accolades {} sont optionnelles) :

```
<schéma> : <information> { ? <requête> } { # <fragment> }
```

- Le **schéma** décrit quelle est la nature de l'information. S'il s'agit d'un numéro de téléphone, alors le schéma sera « **tel** », s'il s'agit d'un site internet, alors le schéma sera « **http** », etc.
- **L'information** est la donnée en tant que telle. Cette information respecte aussi une syntaxe qui dépend du schéma. Ainsi, pour un numéro de téléphone, on peut insérer le numéro tel:0606060606, mais pour des coordonnées GPS il faudra séparer la latitude de la longitude à l'aide d'une virgule geo:123.456789,-12.345678. Pour un site internet, il s'agit d'un chemin hiérarchique.
- **La requête** permet de fournir une précision par rapport à l'information.
- **Le fragment** permet enfin d'accéder à une sous-partie de l'information.

Pour créer un objet URI, utiliser la méthode statique Uri.parse(String uri). Par exemple, pour envoyer un SMS à une personne, on utilise l'URI :

```
Uri sms = Uri.parse("sms:0606060606");
```

Mais je peux aussi indiquer plusieurs destinataires et un corps pour ce message :

Uri sms = Uri.parse("sms:0606060606,0606060607?body=Exemple%20de%diffusion%20de%20mesage");

Comme vous pouvez le voir, le contenu de la chaîne doit être encodé (%20 au lieu d'espace), sinon on rencontrera des problèmes.

### II.2.3. Exemple

Pour créer un intent qui va ouvrir le composeur téléphonique avec le numéro de téléphone 0606060606, on peut utiliser le code suivant [2]:

```
mPasserelle.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    Uri telephone = Uri.parse("tel:0606060606");
    Intent secondeActivite = new Intent(Intent.ACTION_DIAL, telephone);
    startActivity(secondeActivite);
  }
});
```

L'exécution donne le résultat suivant :


**Remarque :**

Encas de l'existence de plusieurs applications, une boite de dialogue s'affiche offrant à l'utilisateur le choix de l'application à utiliser pour exécuter l'action (Viber, Skype pour l'action ACTION_DIAL)

## III. Passage et récupération de paramètres

### III.1. La méthode startActivity(Intent i)

startActivity(Intent i) est une méthode publique de la classe Context. Comme la classe Activity est une sous classe de Context, on peut appeler startActivity(…) directement dans une méthode de la classe Activity ainsi que toutes ses sous classes.

Lorsqu'une activité « A » lance une deuxième activité « B » avec startActivity(…), l'activité « B » se lance et ne retourne aucune information à l'activité « A » lorsqu'elle se termine. Donc cette méthode s'utilise lorsque l'activité « A » n'a aucun besoin des données de l'activité « B ».

**Exemple :** Dans le package com.gest, il existe deux activités « MainActivity » et « Ajout », pour lancer l'activité « Ajout » à partir de « MainActivity » utiliser le code suivant :

```
Intent i = new Intent(MainActivity.this, Ajout.class);
startActivity(i) ;
Remarque : Il faut déclarer la deuxième activité dans AndroidManifest.xml par l'ajout de :
<activity android:name="com.gest.Ajout"></activity> dans <application></application>
```

### III.2. Paramètre simple

#### III.2.1. Introduction

Un paramètre est dit simple lorsque son type est simple. Les types simples sont : int, byte, char, CharSequence, float, short, boolean, double, long, String.

Exemple :

int i = 10 ;

float f = 12.897f ;

String s= "TexteS" ;

boolean b= true ;
CharSequence cs= "TexteCS" ;

### III.2.1. Solution1 : Utilisation de la classe Intent

Pour le passage d'un paramètre simple, la classe Intent possède un ensemble de méthodes qui permettent de passer un paramètre simple, le choix de la méthode à utiliser dépond du type du paramètre à passer.

Pour la lecture de ce paramètre passé, la classe Intent possède un ensemble de méthodes qui permettent de lire ce paramètre. Le choix de la méthode à utiliser dépond du type du paramètre passé :

Le tableau suivant contient ces méthodes :

| Type | Méthode de passage du paramètre | Méthode de lecture du paramètre passé |
|---|---|---|
| int | Intent putExtra(String name, int value) | int getIntExtra(String name, int defaultValue) |
| byte | Intent putExtra(String name, byte value) | byte getByteExtra(String name, byte defaultValue) |
| char | Intent putExtra(String name, char value) | char getCharExtra(String name, char defaultValue) |
| CharSequence | Intent putExtra(String name, CharSequence value) | CharSequence getCharSequenceExtra(String name) |
| float | Intent putExtra (String name, float value) | float getFloatExtra(String name, float defaultValue) |
| short | Intent putExtra(String name, byte value) | short getShortExtra(String name, short defaultValue) |
| boolean | Intent putExtra(String name, boolean value) | boolean getBooleanExtra(String name, boolean defaultValue) |
| double | Intent putExtra(String name, double value) | double getDoubleExtra(String name, double defaultValue) |
| long | Intent putExtra(String name, long value) | long getLongExtra(String name, long defaultValue) |
| String | Intent putExtra(String name, String value) | String getStringExtra(String name) |

### III.2.2. Solution2 : Utilisation de la classe Intent et de la classe Bundle

Dans cette solution, on utilise la méthode « Intent putExtras(Bundle b) » de la classe Intent pour passer un paramètre de type Bundle et « Bundle getExtras() » pour récupérer le Bundle passé en paramètre. Un Intent ne peut passer qu'un seul Bundle.

Un Bundle (paquet en français) est une classe qui permet de stocker un ensemble de paires Clé/Valeur avec la clé est de type String.

Pour le passage d'un paramètre simple, la classe Bundle possède un ensemble de méthodes qui permettent de passer un paramètre simple, le choix de la méthode à utiliser dépond du type du paramètre à passer.

Pour la lecture de ce paramètre, la classe Bundle possède un ensemble de méthodes qui permettent de lire un paramètre simple, le choix de la méthode à utiliser dépond du type du paramètre passé :

Le tableau suivant contient ces méthodes :

| Type | Méthode de passage du paramètre | Méthode de lecture du paramètre passé |
|---|---|---|
| int | void putInt(String key, int value) | int getInt(String key)<br>int getInt(String key, int defaultValue) |
| byte | void putByte(String key, byte value) | byte getByte(String key)<br>byte getByte(String key, byte defaultValue) |
| char | void putChar(String key, char value) | char getChar(String key) |

| | | |
|---|---|---|
| | | char getChar(String key, char defaultValue) |
| CharSequence | void putCharSequence(String key, CharSequence value) | CharSequence getCharSequence(String key)<br>CharSequence getCharSequence(String key, CharSequence defaultValue) |
| float | void putFloat(String key, float value) | float getFloat(String key)<br>float getFloat(String key, float defaultValue) |
| short | void putShort(String key, short value) | short getShort(String key)<br>short getShort(String key, short defaultValue) |
| boolean | void putBoolean(String key, boolean value) | boolean getBoolean(String key)<br>boolean getBoolean(String key, boolean defaultValue) |
| double | void putDouble(String key, double value) | double getDouble(String key)<br>double getDouble(String key, double defaultValue) |
| long | void putLong(String key, long value) | long getLong(String key)<br>long getLong(String key, long defaultValue) |
| String | void putString(String key, String value) | String getString(String key)<br>String getString(String key, String defaultValue) |

### III.2.3. Exemple1 : Utilisation de Solution1

```
public class MainActivity extends Activity {
    protected void passer1() {
            Intent i = new Intent(this, App1.class);
            i.putExtra("texte", "Param Texte");
            i.putExtra("bool", true);
            i.putExtra("val", 15);
            startActivity(i);
    }
}
public class App1 extends Activity {
    private void recupererParam1() {
            Intent i=getIntent();
            String t = i.getStringExtra("texte");
            Boolean b=i.getBooleanExtra("bool",false);
            int max=i.getIntExtra("val",100);
    }
}
```

### III.2.4. Exemple2 : Utilisation de Solution2

```
public class MainActivity extends Activity {
    protected void passer2() {
            Intent i = new Intent(this, App2.class);
            Bundle b = new Bundle();
            b.putString("texte", "Param Texte");
            b.putBoolean("bool", true);
            b.putInt("val", 100);
            i.putExtras(b);
            startActivity(i);
    }
}
```

```
public class App2 extends Activity {
    private void recupererParam1() {
            Intent i = getIntent();
            Bundle bundle = i.getExtras();
            String t = bundle.getString("texte");
            Boolean b=bundle.getBoolean("bool");
            int max=bundle.getInt("val");
    }
}
```

## III.3. Tableau de données simple

### III.3.1. Solution1 : Utilisation de la classe Intent

Comme pour les types simples, la classe Intent propose des méthodes pour passer et récupérer des paramètres de type tableaux de type simple. Les méthodes à utiliser sont de la forme « Intent putExtra(String name, Type[] t) » pour le passage et « Type[] getTypeArrayExtra(Strig name) » pour sa récupération.

### III.3.2. Solution2 : Utilisation de la classe Intent et de la classe Bundle

Comme pour les types simples il faut passer le Bundle dans l'Intent. Par la suite il faut utiliser des méthodes de la classe Bundle de la forme « void putTypeArray(String key, Type[] t) » pour le passage d'un tableau de type simple et des méthodes de la forme « Type[] getTypeArray(String key) » pour sa récupération.

### III.3.3. Exemple

```
public class MainActivity extends Activity {
    private final int[] t= new int[]{15,145,145,225,659,5254,985,8841,554};
    protected void passer() {
            Intent i = new Intent(this, Affichage.class);
            b.putExtra("t", t);
            startActivity(i);
    }
}
public class Affichage extends Activity {
private void afficher() {
            Intent i = getIntent();
            int[] t = bundle.getIntArrayExtra("t");
            for (int j : t) {
                    Log.i("Tab",j) ;;
            }
    }
}
```

## III.4. Paramètre Complexe

### III.4.1. Introduction

Les types complexes sont les types qui ne sont pas simples. Généralement, ils sont définis par le programmeur. Par exemple : Personne, Matière, Voiture, Pays…

Les solutions précédentes ne permettent pas de passer un paramètre de type complexe. Cependant, il existe deux solutions possibles :

- Utiliser l'interface Serializable de Java
- Utiliser l'interface Parcelable d'Android

### III.4.2. Solution1 : Implémenter l'interface Serializable

Dans cette solution, il faut :

- Que la classe de l'objet à passer en paramètre implémente l'interface Serializable,
- Utiliser la méthode « Intent putExtra(String name, Serializable value) » de la classe Intent ou la méthode « void putSerializable(String key, Serializable value) » de la classe Bundle pour passer un objet serialisable,
- Utiliser la méthode « Serializable getSerializableExtra (String name) » de la classe Intent ou la méthode « Serializable getSerializable(String key) » de la classe Bundle pour récupérer un objet serialisable,

### III.4.3. Solution 2 : Implémenter l'interface Parcelable

Dans cette solution, il faut :

- Que la classe de l'objet à passer en paramètre implémente l'interface Parcelable, et redéfinit les deux méthodes abstraites : « int describeContents() » et « void writeToParcel(Parcel dest, int flags) » :
  - int describeContents(), qui permet de définir si vous avez des paramètres spéciaux dans votre Parcelable,
  - void writeToParcel(Parcel dest, int flags), avec dest le Parcel dans lequel nous allons insérer les attributs de notre objet et flags un entier qui vaut la plupart du temps 0. C'est dans cette classe que nous allons écrire dans le Parcel qui transmettra le message.
- Que la classe de l'objet à passer en paramètre ajoute un constructeur qui prend en paramètre un « Parcel » (voir l'exemple 2),
- Que la classe de l'objet à passer en paramètre ajoute une constante : « public static final Parcelable.Creator<Personne> CREATOR » (voir l'exemple 2),
- Utiliser la méthode « Intent putExtra (String name, Parcelable value) » de la classe Intent ou la méthode « void putParcelable(String key, Parcelable value) » de la classe Bundle pour passer un objet parcelable,
- Utiliser la méthode «Parcelable getParcelableExtra(String name) » de la classe Intent ou la méthode « Parcelable getParcelable (String key) » de la classe Bundle pour récupérer un objet parcelable,
- Utiliser la méthode « intent putExtra(String name, Parcelable[] value) » de la classe Intent ou la méthode « putParcelableArray(String key, Parcelable[] value) » de la classe Bundle pour passer un tableau d'objets parcelables,
- Utiliser la méthode « Parcelable[] getParcelableArrayExtra(String name) » de la classe Intent ou la méthode « Parcelable[] getParcelableArray(String key) » de la classe Bundle pour récupérer un tableau d'objets parcelables,

### III.4.4. Exemple1 : Utilisation de l'interface Serializable

```
public class Personne implements Serializable {
    private String nom;
    public Personne(String nom) {
             this.nom = nom;
    }
    public String getNom() {
             return nom;
    }
}
public class MainActivity extends Activity {
    private void passer() {
       Intent i = new Intent(this, AffichagePersonne.class);
       b.putExtra("p",new Personne("Mohamed"));
       startActivity(i);
    }
}
public class AffichagePersonne extends Activity {
    private void afficher() {
             Intent i = getIntent();
             Personne p = (Personne) bundle.getSerializable("p");
             String message = "Nom: " + p.getNom()+"\n";
             Log.i("Personne",message) ;

    }
}
```

### III.4.5. Exemple2 : Utilisation de l'interface Parcelable

```
public class Personne implements Parcelable {
    private String nom;
    public Personne(String nom) {
             this.nom = nom;
    }
    public String getNom() {
             return nom;
    }
   @Override
    public int describeContents() {
```

```
                return 0;
        }

        @Override
        public void writeToParcel(Parcel dest, int flags) {
                dest.writeString(nom);
        }
        public Personne(Parcel in) {
                this.nom = in.readString();
        }
        public static final Parcelable.Creator<Personne> CREATOR = new
        Parcelable.Creator<Personne>() {
                @Override
                public Personne createFromParcel(Parcel source) {
                        return new Personne(source);
                }
                @Override
                public Personne[] newArray(int size) {
                        return new Personne[size];
                }
        };
}
public class MainActivity extends Activity {
        private void passer() {
           Intent i = new Intent(this, AffichagePersonne.class);
           b.putExtra("p",new Personne("Mohamed"));
           startActivity(i);
        }
}
public class AffichagePersonne extends Activity {
        private void afficher() {
                Intent i = getIntent();
                Personne p = (Personne) bundle.getParcelableExtra("p");
                String message = "Nom: " + p.getNom()+"\n";
                Log.i("Personne",message) ;
        }
}
```

# IV. Récupération d'un résultat

## IV.1. La méthode startActivityForResult()

startActivityForResult(Intent i,int requestCode) est une méthode publique de la classe Context. Comme la classe Activity est une sous classe de Context, on peut appeler startActivityForResult(…) directement dans une méthode de la classe Activity ainsi que toutes ses sous classes.

Lorsqu'une activité « A » lance une deuxième activité « B » avec startActivityForResult(…), l'activité « B » se lance et retourne un feedback à l'activité « A » lorsqu'elle se termine.

Le requestCode permet à la classe « A » d'identifier qui est l'activité qui vient de se terminer. Ceci est utile lorsqu'une activité lance plusieurs activités avec startActivityForResult(…).

**Exemple :** Dans le package com.gest, il existe deux activités « MainActivity » et « Param », pour lancer l'activité « Param » à partir de « MainActivity » utiliser le code suivant :

```
Intent i = new Intent(MainActivity.this, Ajout.class);
startActivityForResult(i,13) ;
Remarque : Il faut déclarer la deuxième activité dans AndroidManifest.xml par l'ajout de :
<activity android:name="com.gest.Param"></activity> dans <application></application>
```

## IV.2. Les méthodes setResult(int resultCode) et setResult(int resultCode, Intent data)

La classe activité contient les deux constantes RESULT_OK et RESULT_CANCELED qui permettent d'indiquer si l'activité s'est terminée normalement (OK) ou l'utilisateur l'a abandonnée (CANCELED).

Si l'activité appelée n'a pas de données à transmettre à l'activité appelante alors elle utilise la méthode setResult(int resultCode) avant d'appeler finish(). Exemple :

```
setResult(RESULT_OK);
finish() ;
```

Si l'activité appelée a des données à transmettre à l'activité appelante alors elle utilise la méthode setResult(int resultCode,Intent data) avant d'appeler finish(). Exemple :

```
Intent i = new Intent() ;
i.putExtra("nom","Mohamed") ;
setResult(RESULT_OK,i);
finish() ;
```

### IV.3. La méthode onActivityResult()

Quand l'activité appelée s'arrêtera, la première méthode de *callback* appelée dans l'activité précédente sera void onActivityResult(int requestCode, int resultCode, Intent data). On retrouve requestCode, qui sera le même code que celui passé dans le startActivityForResult et qui permet de repérer quel intent a provoqué l'appel de l'activité dont le cycle vient de s'interrompre. resultCode est quant à lui un code renvoyé par l'activité qui indique comment elle s'est terminée (typiquement Activity.RESULT_OK si l'activité s'est terminée normalement, ou Activity.RESULT_CANCELED s'il y a eu un problème ou qu'aucun code de retour n'a été précisé). Enfin, data est un intent qui contient éventuellement des données [2].

### IV.4. Exemple

```
public class MainActivity extends Activity {
    private void afficherA() {
       Intent i = new Intent(this, A.class);
       startActivityForResult(i,11);
    }
    private void afficherB() {
       Intent i = new Intent(this, B.class);
       startActivityForResult(i,12);
    }
@Override
 protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
             case 11 :
                      if (resultCode == RESULT_OK) {
                               String nom=data.getStringExtra("nom");
                               Log.i("Nom :",nom );
                      }
                      break ;
             case 12 :
                      if (resultCode == RESULT_OK) {
                               String prenom=data.getStringExtra("prenom ");
                               Log.i("Prénom :", prenom);
                      }
                      break ;

    }
  }
}
public class A extends Activity {
    private void valider() {
             Intent i = new Intent() ;
             i.putExtra("nom","Mohamed") ;
             setResult(RESULT_OK,i) ;
             finish() ;
    }
    private void annuler() {
             setResult(RESULT_CANCELED) ;
             finish() ;
    }
}
public class B extends Activity {
    private void valider() {
             Intent i = new Intent() ;
             i.putExtra("prenom","Saleh") ;
             setResult(RESULT_OK,i) ;
           finish() ;
    }
    private void annuler() {
             setResult(RESULT_CANCELED) ;
             finish() ;
    }
}
```

# Références


[1]http://campus.hesge.ch/Daehne/2013-2014/Module635.1/Prog/Documents/05-CommunicationComposants.pdf

[2] https://openclassrooms.com/courses/creez-des-applications-pour-android/la-communication-entre-composants