Android – Activités et Navigation

1. Les activités

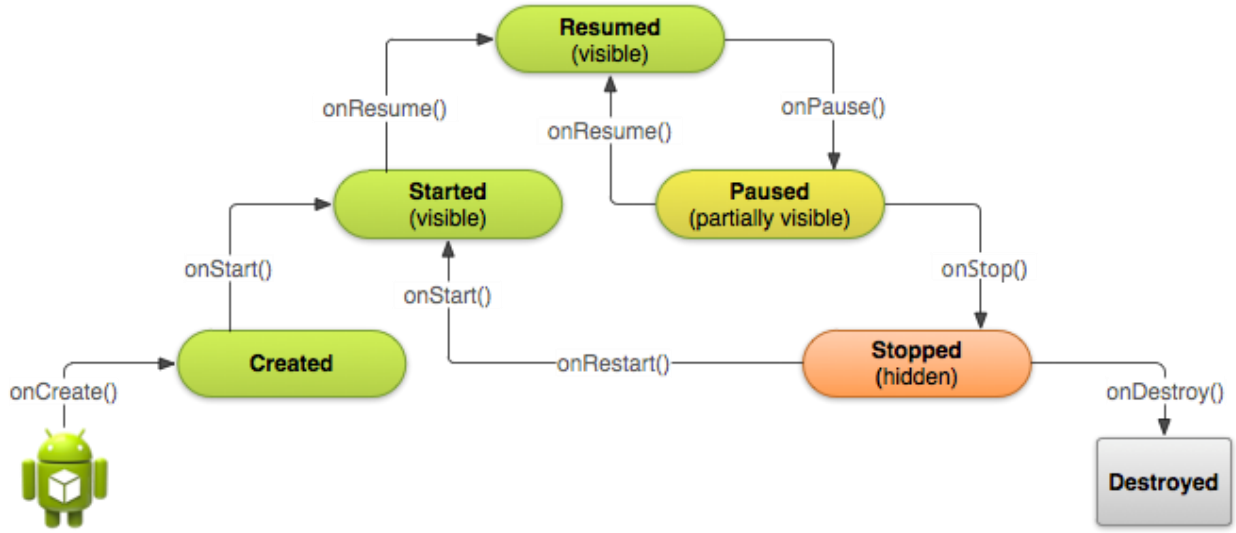
Nous avons vu que tout écran présenté à l'utilisateur est porté par une activité. Une activité occupe tout l'écran du terminal, l'application ne peut donc en afficher qu'une à la fois. Il faut donc prévoir un moyen de naviguer entre les différentes applications. *C'est l'objectif de ce chapitre.*

Lors de la création d'un projet, Android Studio crée, par défaut, un fichier MainActivity.java qui correspond au fichier activity_main.xml qui définit l'interface graphique de l'activité.

Les activités héritent de la classe android.app.Activity ou d'une classe dérivée (par exemple : AppCompatActivity). On ne déclare pas de constructeur pour une activité. Par contre, il faut suivre le cycle de vie d'une activité et surcharger les méthodes correspondantes aux différents états de ce cycle de vie.

1.1.Cycle de vie d'une activité :

Le cycle de vie d'une activité peut être représenté par le schéma suivant qui montre la succession des différents états logiques de cette dernière.



`onCreate()` : appelée au lancement de l'activité (sa création) ;
`onStart()` : appelée après la création de l'activité (`onCreate`) ou sa recréation (`onRestart`), ce qui signifie le lancement effectif de l'activité (elle est maintenant visible).
`onResume()` est appelée lorsque l'activité est présentée à l'utilisateur (1er plan). Elle est appelée pour exécuter tous les traitements nécessaires au fonctionnement de l'activité, initialiser des variables et les écouteurs. Ces traitement devront être arrêtés lors de l'appel de la méthode `onPause`() et relancés si besoin lors d'un futur appel à `onResume`(). Après ces trois appels, l'activité est utilisable et peut recevoir les interactions de l'utilisateur.
`onPause()` : appelée lorsqu'une autre activité passe au 1er plan. C'est le pendant de la méthode `onResume()`. Juste avant l'appel de cette méthode, la méthode `onSaveInstanceState`() est appelée afin de sauvegarder les informations portées par l'activité. Elle arrête tous les traitements effectués par l'activité.
`onStop()` : appelée lorsque l'activité n'est plus présentée à l'utilisateur.
`onDestroy()` : appelée lorsque l'activité est détruite, soit par l'appel à la méthode `finish()`, soit directement par le système pour libérer les ressources. L'état est sauvegardé à l'aide de la méthode `onSaveInstanceState()`. Cet état est transmis en paramètre à la méthode `onCreate()`, ce qui permet de la restaurer. Les sauvegardes et restaurations de l'état des vues des layouts sont effectuées par le système Android, à condition que chaque vue ait son propre ID dans le layout.

En résumé, quand l'activité n'est plus affichée, les méthodes `onResume()` et `onStop()` sont appelées. Lorsqu'elle est de nouveau affichée, `onRestart()`, `onStart()` et `onResume()` sont appelées.

Le changement de configuration ou d'orientation de l'écran provoque un redémarrage de l'activité, il faut alors sauvegarder et restaurer l'état de l'activité.

La classe qui correspond à l'activité peut redéfinir chaque méthode onEvenement(). Il est impératif, lors de la surcharge, d'appeler explicitement la méthode de la classe mère.

1.2. Initialisation de l'activité

La méthode `onCreate()` est appelée au début du cycle de vie d'une activité, et n'est appelée qu'une seule fois. Elle reçoit en paramètre un objet de type Bundle qui a pour rôle de stocker des données temporaires.

```
protected void onCreate(Bundle savedInstanceState)
```

- La méthode `onCreate()` va lier le code Java avec le fichier de layout XML qui décrit l'interface de l'activité. Cette liaison est effectuée avec la méthode setContentView de la classe Activity :

  ```
  setContentView(R.layout.identifiant);
  ```

  Le paramètre de type `int` passé en argument de la méthode correspond à l'identifiant de la ressource (fichier layout XML) généré automatiquement dans le répertoire `/res/layout`. Le fichier étant une ressource de type `layout`, l'identifiant est de la forme `R.layout.identifiant`. (identifiant sera le nom du fichier XML).
- Outre la liaison du fichier de layout à l'activité, il faut obtenir une référence sur chacun des composants qui constituent l'interface pour pouvoir les manipuler dans l'activité. La classe `Activity` dispose de la méthode

  ```
  View findViewById(R.id.nomComposant);
  ```

  La méthode prend en paramètre l'identifiant du composant souhaité et renvoie un objet de type View qu'il faudra convertir en composant typé pour accéder à ses propres méthodes. *Par exemple :*

  ```
  EditText etNbPizzas = (EditText) findViewById(R.id.etNbPizzas);
  ```

1.3. Inscription dans le Manifest

Toutes les activités doivent être déclarées dans le fichier : AndroidManifest.xml.
La déclaration d'une activité dans ce fichier est réalisée par la balise <activity> dans la section <application>. Le seul attribut obligatoire est name qui doit contenir comme valeur le nom complet de la classe Java de l'activité. Si le nom du package est le même que celui de l'application, on peut le substituer par '.'.

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

Les filtres d'intention servent à indiquer à une activité leurs comportements ou le type d'intents qu'ils peuvent implicitement traiter. Tout composant souhaitant être prévenu par des intentions doit déclarer des filtres d'intention.

Dans l'exemple ci-dessous, les filtres d'intent indiquent au système l'activité à lancer au démarrage de l'application. Ces deux filtres d'intention signifient respectivement que l'activité :

- est la principale de l'application, c'est le rôle de la balise <action>
- appartient à la catégorie Launcher (balise <category>), c'est-à-dire qu'elle sera disponible en raccourci depuis le menu principal de lancement d'application d'Android.

Par exemple :

```
<activity android:name=".ActivitePrincipale">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
</activity>
```

1.4. Back stack

Un écran Android ne présente qu'une vue à la fois. La succession des interfaces est stockée dans une pile appelée back stack, ce qui permet de revenir à l'écran précédent à l'aide du bouton « retour » du téléphone.
Toutes les activités lancées sont stockées dans la cette pile. Quand une nouvelle activité est lancée, elle est ajoutée en haut de la back stack. Quand le bouton « retour » est sollicité, l'activité en cours est fermée et celle se trouvant juste en-dessous dans la back list est ouverte.

2. Les Intentions

Une application Android est composée de plusieurs activités. Il faut donc prévoir le passage d'une activité à une autre, avec la possibilité d'envoyer des données entre les activités (envoi/retour).

2.1. Activité fille, intention et paramètre

Supposons qu'on souhaite le déroulement suivant de l'application :

Le menu New => Activity => Empty Activity permet de créer l'activité fille (activité secondaire). On retrouve la description de cette activité fille et la relation avec l'activité principale dans le fichier **AndroidManifest.xml** :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="scootspizza.fr.scootmobv2">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".ActivitePrincipale">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name=".ActiviteSecondaire"></activity>
    </application>

</manifest>
```

Description de l'activité principale : **activity_activite_principale.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="scootspizza.fr.scootmobv2.ActivitePrincipale">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:textStyle="bold"
        android:text="@string/label_et_nb_pizzas" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/idEtNbPizzas" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:gravity="right">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/label_bt_envoyer"
            android:id="@+id/idBtEnvoyer"
            android:onClick="envoyer"/>
    </LinearLayout>
</LinearLayout>
```

Le fichier **strings.xml** contient, quant à lui, les déclarations utilisées dans les balises @string dans les deux fichiers xml précédents :

```
<resources>
    <string name="app_name">ScootMobv2</string>
    <string name="label_et_nb_pizzas">Saisir le nombre de pizzas : </string>
    <string name="label_nb_pizzas_envoyees"> Nombre de pizzas saisies : </string>
    <string name="label_bouton_envoyer">Envoyer</string>
    <string name="label_bouton_retourner">Retourner</string>
</resources>
```

Le fichier **ActivitePrincipale.java** correspondant :

```
public class ActivitePrincipale extends AppCompatActivity {
    EditText etNbPizzas ;
    Button btEnvoyer;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_activite_principale);

        etNbPizzas = (EditText) findViewById(R.id.idEtNbPizzas);
        btnEnvoyer = (Button) findViewById(R.id.idBtEnvoyer);
    }
```

```
    public void envoyer(View vue) {
        String nbPizzasSaisies = idEtNbPizzas.getText().toString();

        Bundle paquet = new Bundle();
        paquet.putString("nbPizzas", nbPizzasSaisies);

        Intent intentEnvoyer = new Intent(this, ActiviteSecondaire.class);
        intentEnvoyer.putExtras(paquet);
        startActivity(intentEnvoyer);
    }
}
```

Lancement de l'activité

Représente une intention, ici le lancement d'une nouvelle activité

La description de l'activité secondaire est définie dans le fichier **activity_activite_secondaire.xml** :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="scootspizza.fr.scootmobv2.ActiviteSecondaire">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/tvNbPizzas"
        android:textSize="18sp"
        android:textStyle="bold"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:gravity="right">
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/label_bouton_retourner"
            android:id="@+id/btnRetourner"
            android:onClick="retourner"/>
    </LinearLayout>
</LinearLayout>
```

Le fichier **ActiviteSecondaire.java** correspondant :

```
public class ActiviteSecondaire extends AppCompatActivity {
    TextView tvNbPizzas;
    Button btnRetourner;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_activite_secondaire);

        tvNbPizzas = (TextView) findViewById(R.id.tvNbPizzas);

        Bundle paquet = this.getIntent().getExtras();
        String nbPizzas = paquet.getString("nbPizzas");

        tvNbPizzas.setText("Nombre de pizzas envoyées : "+nbPizzas);
    }
```

```
    public void retourner(View vue) {
        Intent intentRetourner = new Intent(this,ActivitePrincipale.class);
        startActivity(intentRetourner);
    }
}
```

Le lancement d'une activité se fait par un appel à la méthode **`startActivity`** de la classe Activity.

```
public void startActivity (Intent intent)
```

L'objet `Intent` passé en paramètre représente une intention, c'est-à-dire une action devant être exécutée par le système. Si l'action consiste à lancer une nouvelle activité, il faut utiliser le constructeur de la classe `Intent` :

```
public Intent(Context packageContext, Class< ?> classe)
```

- Le 1er paramètre représente le contexte d'utilisation.
- Le 2ème paramètre représente la classe de l'activité à lancer.

Dans notre exemple, cela correspond :
*Dans **ActivitePrincipale.java**, dans la méthode envoyer() :*

```
Intent intentEnvoyer = new Intent(this, ActiviteSecondaire.class);
intentEnvoyer.putExtras(paquet);
startActivity(intentEnvoyer);
```

*et dans **ActiviteSecondaire.java**, dans la méthode retourner() :*

```
Intent intentRetourner = new Intent(this,ActivitePrincipale.class);
startActivity(intentRetourner);
```

2.2. Passage de paramètres

Pour passer des données d'une activité à une autre, on utilise la capacité de stockage de l'objet `Intent`. Ces données peuvent être :

- stockées au préalable dans un objet de type `Bundle`, sous forme de couple clé/valeur puis envoyées par l'objet `Intent`
- envoyées par l'objet `Intent` sans l'intermédiaire de `Bundle` via la méthode `putExtra`()

Utilisation de la méthode `putExtras(Bundle)` de la classe `Intent` :
Dans notre cas, nous envoyons à l'activité secondaire la valeur saisie dans la zone de texte. D'où les instructions suivantes dans ActivitePrincipale.java :

```
Bundle paquet = new Bundle();
paquet.putString("nbPizzas", nbPizzasSaisies);
Intent intentEnvoyer = new Intent(this, ActiviteSecondaire.class);
intentEnvoyer.putExtras(paquet);
startActivity(intentEnvoyer);
```

Utilisation de la méthode putExtra(name, value) de la classe Intent :
La classe Intent dispose de la méthode putExtra, déclinée pour chaque type de données à stocker :

```
public Intent putExtra(String name, boolean value)
public Intent putExtra(String name, boolean[] value)
public Intent putExtra(String name, int value)
public Intent putExtra(String name, int[] value)
public Intent putExtra(String name, String value)
public Intent putExtra(String name, String[] value)
public Intent putExtra(String name, Parcelable value)
public Intent putExtra(String name, Parcelable[] value)
public Intent putExtra(String name, Serializable value)
…
```

Il est donc possible de remplacer le code précédent en utilisant la méthode `putExtra()` de la classe `Intent` sans passer par l'objet `Bundle` :

```
Intent intentEnvoyer = new Intent(this, ActiviteSecondaire.class);
intentEnvoyer.putExtra("nbPizzas",nbPizzasSaisies);
startActivity(intentEnvoyer);
```

2.3. Passage de paramètres de type objets complexes

Pour passer des objets complexes d'une activité à une autre, il faut sérialiser ces objets. La sérialisation est un mécanisme introduit en Java qui permet de rendre un objet persistant pour le stockage ou l'envoi à travers le réseau. La dé-sérialisation est le procédé inverse qui permet de retrouver la représentation initiale de l'objet. En programmation Android, il existe deux manières d'effectuer la sérialisation/dé-sérialisation :

- en utilisant la librairie Gson proposé par Google ;
- en implémentant l'interface `Serializable ou Parcelable`.

On va privilégier la librairie Gson. Néanmoins, on va étudier également l'interface `Parcelable` qui est un protocole de sérialisation propre à Android et qui est plus performant que l'interface `Serializable`. La transformation des objets en `Parcel` permettra leur passage entre les activités.
Là encore, on peut utiliser le `Bundle` ou alors utiliser la méthode `putExtra(String name, parcelable value)`. On utilisera cette dernière.

Création de la classe et implémentation de l'interface `Parcelable` :
La classe représentant l'objet implémente l'interface `Parcelable`. Cela rajoutera deux méthodes :

- `describeContents()` : décrit le contenu de l'objet `Parcel` et plus précisément le nombre d'objets spéciaux contenus dans `Parcel`.
- `writeToParcel()` : écrit les champs de l'objet dans un `Parcel`. C'est l'opération de « sérialisation ».

Exemple : nous allons créer une activité principale qui permet à un livreur de saisir son nom. Cette activité va renvoyer les données concernant le livreur qui a saisi son nom à l'activité secondaire qui les affichera.

```
public class Livreur implements Parcelable {

    private String matricule;
    private String nom;
    private String prenom;

    public Livreur (String matricule, String nom, String prenom) {

        this.matricule = matricule;
        this.nom = nom;
        this.prenom = prenom;
    }

@Override
public int describeContents() {
    return 0;
}

@Override
public void writeToParcel(Parcel dest, int flags) {
    dest.writeString(matricule);
    dest.writeString(nom);
    dest.writeString(prenom);
}
```

La méthode `writeToParcel()` enregistre les différents attributs de la classe dans le Parcel passé en argument à la méthode.

Ensuite, il faut créer un objet `CREATOR` de la classe `Parcelable` :

```
public static final Parcelable.Creator<Livreur>
    CREATOR = new Parcelable.Creator<Livreur>()
{

    public Livreur createFromParcel(Parcel in) {
        return new Livreur(in);
    }

    public Livreur[] newArray(int size) {
        return new Livreur[size];
    }
};
```

Et enfin, il faut créer un constructeur qui permet de reconstruire l'objet à partir d'un `Parcel` (processus de dé-sérialisation) :

```
protected Livreur(Parcel in) {

    this.matricule = in.readString();
    this.nom = in.readString();
    this.prenom = in.readString();

}
```

Utilisation :

Dans l'activité principale :

```
Livreur liv1 = new Livreur("1", "Dupont","Marie");

Intent intentEnvoyer = new Intent(this,MainChildActivity.class);
intentEnvoyer.putExtra("livreur", liv1);
startActivity(intentEnvoyer);
```

Dans l'activité secondaire :

```
Livreur livr1 = this.getIntent().getParcelableExtra("livreur");

tvMatricule.setText(livr1.getMatricule());
tvNom.setText(livr1.getNom());
tvPrenom.setText(livr1.getPrenom());
```

Utilisation de la bibliothèque Gson

La librairie est simple d'utilisation. On utilisera les deux méthodes principales :
toJson() : crée un objet à partir d'un résultat Json et du type de retour désiré ;
fromJson() : convertir un objet au format Json.

⇨ **Mettre en place la librairie**

Pour utiliser la librairie, il faut la mettre dans le fichier build.gradle dans les dependencies et relancer le build :
`Par exemple : implementation 'com.google.code.gson:gson:1.7.2'`

⇨ **Sérialiser avec toJson() : convertir un objet en json.**

```
Gson gson = new GsonBuilder().create();
String jsonCommande = gson.toJson(commande); //commande étant un objet de type Commande
```

⇨ **Désérialiser avec fromJson() : convertir un résultat Json en objet**

```
Gson gson = new GsonBuilder().setPrettyPrinting().create();
Commande commande = gson.fromJson(jsonCommande, Commande.class);
```

En résumé :
Une application Android est constituée de plusieurs activités. La navigation d'une activité à une autre, avec ou sans passage de paramètres, est réalisée à l'aide d'objets Intent.