Android

Applications et activités

Composition d'une application

Une application est composée d'une ou plusieurs *activités*. Chacune gère un écran d'interaction avec l'utilisateur et est définie par une classe Java.

Une application complexe peut aussi contenir :

- des *services* : ce sont des processus qui tournent en arrière-plan,
- des *fournisseurs de contenu* : ils représentent une sorte de base de données.
- des *récepteurs d'annonces* : pour gérer des événements globaux envoyés par le système à toutes les applications.

Déclaration d'une application

Le fichier `AndroidManifest.xml` déclare les éléments d'une application, avec un '.' devant le nom de classe des activités :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ...>
        <activity android:name=".MainActivity"
                  ... />
        <activity android:name=".EditActivity"
                  ... />
        ...
    </application>
</manifest>
```

`<application>` est le seul élément sous la racine `<manifest>` et ses filles sont des `<activity>`.

Sécurité des applications (pour info)

Chaque application est associée à un UID (compte utilisateur Unix) unique dans le système. Ce compte les protège les unes des autres. Cet UID peut être défini dans le fichier AndroidManifest.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...
        android:sharedUserId="ma.ceft.test">...

</manifest>
```

Définir l'attribut `android:sharedUserId` avec une chaîne identique à une autre application, et signer les deux applications avec le même certificat, permet à l'une d'accéder à l'autre.

Autorisations d'une application

Une application doit déclarer les autorisations dont elle a besoin : accès à internet, caméra, carnet d'adresse, GPS, etc.

Cela se fait en rajoutant des élements dans le manifeste :

```
<manifest ... >
    <uses-permission
        android:name="android.permission.INTERNET" />
    ...
</manifest>
```

Consulter cette page pour la liste des permissions existantes.

NB: les premières activités que vous créerez n'auront besoin d'aucune permission.

Démarrage d'une application

L'une des activités est marquée comme démarrable de l'extérieur :

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

Un `<intent-filter>` déclare les conditions de démarrage d'une activité, ici il dit que c'est l'activité principale.

Démarrage d'une activité et `Intents`

Les activités sont démarrées à l'aide d'`Intents`. Un `Intent` contient une demande destinée à une activité, par exemple, composer un numéro de téléphone ou lancer l'application.

- *action* : spécifie ce que l'`Intent` demande. Il y en a de très nombreuses :
  - `VIEW` pour afficher quelque chose, `EDIT` pour modifier une information, `SEARCH`...
- *données* : selon l'action, ça peut être un numéro de téléphone, l'identifiant d'une information...
- *catégorie* : information supplémentaire sur l'action, par exemple, `...LAUNCHER` pour lancer une application.

Une application a la possibilité de lancer certaines activités d'une autre application, celles qui ont un `intent-filter`.

Lancement d'une activité par programme

Soit une application contenant deux activités : `Activ1` et `Activ2`. La première lance la seconde par :

```
Intent intent = new Intent(this, Activ2.class);
startActivity(intent);
```

L'instruction `startActivity` démarre `Activ2`. Celle-ci se met devant `Activ1` qui se met alors en sommeil.

Ce bout de code est employé par exemple lorsqu'un bouton, un menu, etc. est cliqué. Seule contrainte : que ces deux activités soient déclarées dans `AndroidManifest.xml`.

Lancement d'une application Android

Il n'est pas possible de montrer toutes les possibilités, mais par exemple, voici comment ouvrir le navigateur sur un URL :

```
String url ="https://fpt.ac.ma";

intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
startActivity(intent);
```

L'action `VIEW` avec un *URI* (généralisation d'un *URL*) est interprétée par Android, cela fait ouvrir automatiquement le navigateur.

Lancement d'une activité d'une autre application

Soit une seconde application dans le package ma.ceft.appli2.
Une activité peut la lancer ainsi :

```
intent = new Intent(Intent.ACTION_MAIN);
intent.addCategory(Intent.CATEGORY_LAUNCHER);
intent.setClassName(
    "ma.ceft.appli2",
    "ma.ceft.appli2.MainActivity");
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
startActivity(intent);
```

Cela consiste à créer un `Intent` d'action `MAIN` et de catégorie `LAUNCHER` pour la classe `MainActivity` de l'autre application.

Applications

Fonctionnement d'une application

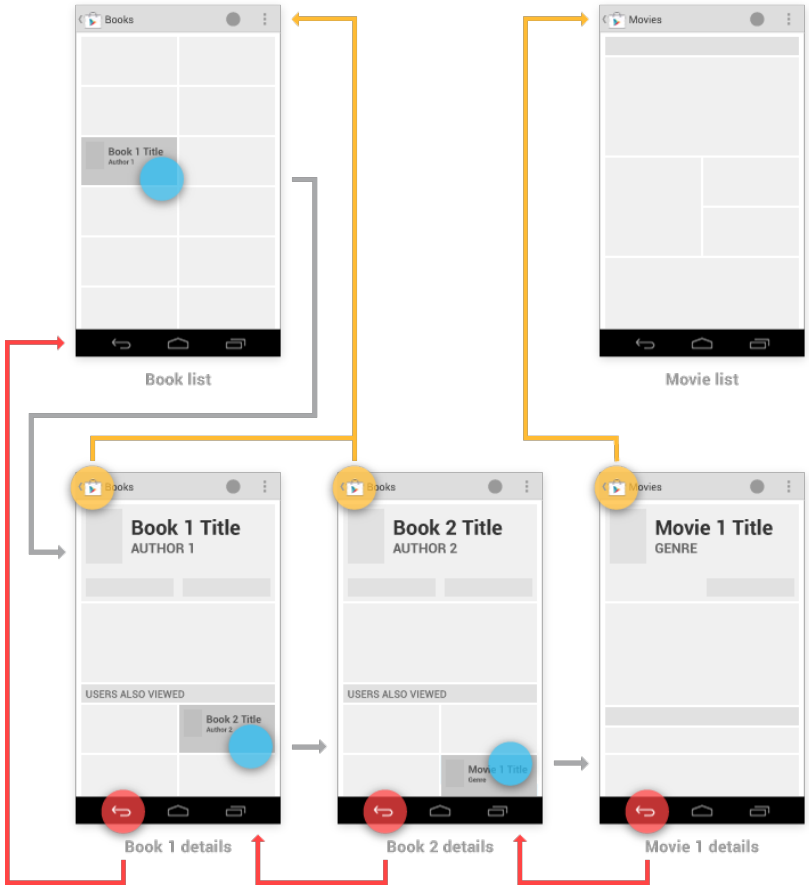
Au début, le système Android lance l'activité qui est marquée action=`MAIN` et catégorie=`LAUNCHER` dans `AndroidManifest.xml`.

Ensuite, d'autres activités peuvent être démarrées. Chacune se met « devant » les autres comme sur une pile. Deux cas sont possibles :

- La précédente activité se termine, on ne revient pas dedans.
  Par exemple, une activité où on tape son login et son mot de passe lance l'activité principale et se termine.
- La précédente activité attend la fin de la nouvelle car elle lui demande un résultat en retour.
  Exemple : une activité de type liste d'items lance une activité pour éditer un item quand on clique longuement dessus, mais attend la fin de l'édition pour rafraîchir la liste.

Navigation entre activités

Voici un schéma (Google) illustrant les possibilités de navigation parmi plusieurs activités.



Lancement sans attente

Rappel, pour lancer `Activ2` à partir de `Activ1` :

```
Intent intent = new Intent(this, Activ2.class);
startActivity(intent);
```

On peut demander la terminaison de `this` après lancement de `Activ2` ainsi :

```
Intent intent = new Intent(this, Activ2.class);
startActivity(intent);
finish();
```

`finish()` fait terminer l'activité courante. L'utilisateur ne pourra pas faire back dessus, car elle disparaît de la pile.

Lancement avec attente de résultat

Le lancement d'une activité avec attente de résultat est plus complexe. Il faut définir un *code d'appel* `RequestCode` fourni au lancement.

```
private static final int APPEL_ACTIV2 = 1;
Intent intent = new Intent(this, Activ2.class);
startActivityForResult(intent, APPEL_ACTIV2);
```

Ce code identifie l'activité lancée, afin de savoir plus tard que c'est d'elle qu'on revient. Par exemple, on pourrait lancer au choix plusieurs activités : édition, copie, suppression d'informations. Il faut pouvoir les distinguer au retour.

Consulter cette page.

Lancement avec attente, suite

Ensuite, il faut définir une méthode *callback* qui est appelée lorsqu'on revient dans notre activité :

```
@Override
protected void onActivityResult(
        int requestCode, int resultCode, Intent data)
{
    // uti a fait back
    if (resultCode == Activity.RESULT_CANCELED) return;
    // selon le code d'appel
    switch (requestCode) {
        case APPEL_ACTIV2:  // on revient de Activ2
                ...
    }
}
```

Terminaison d'une activité

L'activité lancée par la première peut se terminer pour deux raisons :

- Volontairement, en appelant la méthode `finish()` :

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

- À cause du bouton « back » du téléphone, son action revient à faire ceci :

```
setResult(RESULT_CANCELED);
finish();
```

Dans ces deux cas, on revient dans l'activité appelante (sauf si elle-même avait fait `finish()`.

Méthode `onActivityResult`

Quand on revient dans l'activité appelante, Android lui fait exécuter cette méthode :

`onActivityResult(int requestCode, int resultCode, Intent data)`

- `requestCode` est le code d'appel de `startActivityForResult`
- `resultCode` vaut soit `RESULT_CANCELED` soit `RESULT_OK`, voir le transparent précédent
- `data` est fourni par l'activité appelée et qui vient de se terminer.

Ces deux dernières viennent d'un appel à `setResult(resultCode, data)`

Transport d'informations dans un `Intent`

Les `Intent` servent aussi à transporter des informations d'une activité à l'autre : les *extras*.

Voici comment placer des données dans un `Intent` :

```
Intent intent =
    new Intent(this, DeleteInfoActivity.class);
intent.putExtra("idInfo", idInfo);
intent.putExtra("hiddencopy", hiddencopy);
startActivity(intent);
```

`putExtra(nom, valeur)` rajoute un couple (nom, valeur) dans l'intent. La valeur doit être *sérialisable* : nombres, chaînes et structures simples.

Extraction d'informations d'un `Intent`

Ces instructions récupèrent les données d'un `Intent` :

```
Intent intent = getIntent();
Integer idInfo = intent.getIntExtra("idInfo", -1);
bool hidden = intent.getBooleanExtra("hiddencopy", false);
```

- `getIntent()` retourne l'`Intent` qui a démarré cette activité.
- `get`*Type*`Extra(`*nom*, *valeur par défaut*`)` retourne la valeur de ce nom si elle en fait partie, la valeur par défaut sinon.

Il est très recommandé de placer les chaînes dans des constantes, dans la classe appelée :

```
public static final String EXTRA_IDINFO = "idInfo";
public static final String EXTRA_HIDDEN = "hiddencopy";
```

Contexte d'application

Pour finir sur les applications, il faut savoir qu'il y a un objet global vivant pendant tout le fonctionnement d'une application : le contexte d'application. Voici comment le récupérer :

```
Application context = this.getApplicationContext();
```

Par défaut, c'est un objet neutre ne contenant que des informations Android.

Il est possible de le sous-classer afin de stocker des variables globales de l'application.

Définition d'un contexte d'application

Pour commencer, dériver une sous-classe de `Application` :

```
public class MonApplication extends Application
{
    // variable globale de l'application
    private int varglob;

    public int getVarGlob() { return varglob; }

    // initialisation du contexte
    @Override public void onCreate() {
        super.onCreate();
        varglob = 3;
    }
}
```

Définition d'un contexte d'application, suite

Ensuite, la déclarer dans `AndroidManifest.xml`, dans l'attribut `android:name` de l'élément `<application>`, mettre un point devant :

```
<manifest xmlns:android="..." ...>
    <application android:name=".MonApplication"
        android:icon="@drawable/icon"
        android:label="@string/app_name">
        ...
```

Définition d'un contexte d'application, fin

Enfin, l'utiliser dans n'importe laquelle des activités :

```
// récupérer le contexte d'application
MonApplication context =
    (MonApplication) this.getApplicationContext();

// utiliser la variable globale
... context.getVarGlob() ...
```

Remarquez la conversion de type du contexte.

Activités

Présentation

Voyons maintenant comment fonctionnent les activités.

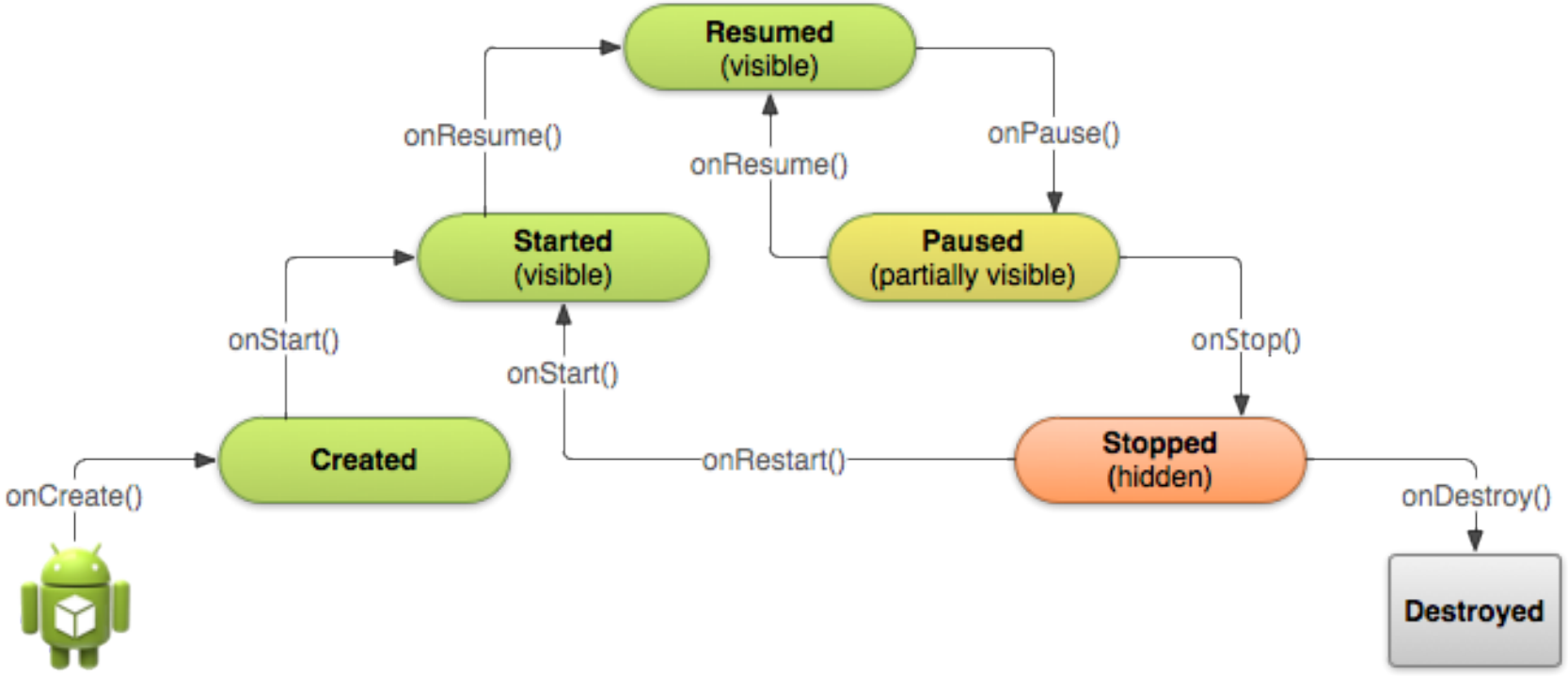
- Démarrage (à cause d'un `Intent`)
- Apparition/masquage sur écran
- Terminaison

Une activité se trouve dans l'un de ces états :

- active (*resumed*) : elle est sur le devant, l'utilisateur peut jouer avec,
- en pause (*paused*) : partiellement cachée et inactive, car une autre activité est venue devant,
- stoppée (*stopped*) : totalement invisible et inactive, ses variables sont préservées mais elle ne tourne plus.

Cycle de vie d'une activité

Ce diagramme résume les changement d'états d'une activité :



Événements de changement d'état

La classe `Activity` reçoit des événements de la part du système Android, ça appelle des fonctions appelées *callbacks*.

Exemples :

`onCreate` Un `Intent` arrive dans l'application, il déclenche la création d'une activité, dont l'interface.

`onPause` Le système prévient l'activité qu'une autre activité est passée devant, il faut enregistrer les informations au cas où l'utilisateur ne revienne pas.

Squelette d'activité

```
public class EditActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState) {
        // obligatoire
        super.onCreate(savedInstanceState);

        // met en place les vues de cette activité
        setContentView(R.layout.edit_activity);
    }
}
```

`@Override` signifie que cette méthode remplace celle héritée de la superclasse. Il faut quand même l'appeler sur `super` en premier.

Terminaison d'une activité

Voici la prise en compte de la terminaison définitive d'une activité, avec la fermeture d'une base de données :

```
@Override
public void onDestroy() {
    // obligatoire
    super.onDestroy();

    // fermer la base
    db.close();
}
```

Pause d'une activité

Cela arrive quand une nouvelle activité passe devant, exemple : un appel téléphonique. Il faut libérer les ressources qui consomment de l'énergie (animations, GPS...).

```
@Override public void onPause() {
    super.onPause();
    // arrêter les animations sur l'écran
    ...
}
@Override public void onResume() {
    super.onResume();
    // démarrer les animations
    ...
}
```

Arrêt d'une activité

Cela se produit quand l'utilisateur change d'application dans le sélecteur d'applications, ou qu'il change d'activité dans votre application. Cette activité n'est plus visible et doit enregistrer ses données.

Il y a deux méthodes concernées :

- `protected void onStop()` : l'application est arrêtée, libérer les ressources,
- `protected void onStart()` : l'application démarre, allouer les ressources.

Il faut comprendre que les utilisateurs peuvent changer d'application à tout moment. La votre doit être capable de résister à ça.

Enregistrement de valeurs d'une exécution à l'autre

Il est possible de sauver des informations d'un lancement à l'autre de l'application (certains cas comme la rotation de l'écran ou une interruption par une autre activité), dans un `Bundle`. C'est un container de données quelconques, sous forme de couples ("nom", valeur).

```
static final String ETAT_SCORE = "ScoreJoueur"; // nom
private int mScoreJoueur = 0;                    // valeur

@Override
public void onSaveInstanceState(Bundle etat) {
    // enregistrer l'état courant
    etat.putInt(ETAT_SCORE, mScoreJoueur);
    super.onSaveInstanceState(etat);
}
```

Restaurer l'état au lancement

La méthode `onRestoreInstanceState` reçoit un paramètre de type `Bundle` (comme la méthode `onCreate`, mais dans cette dernière, il peut être `null`). Il contient l'état précédemment sauvé.

```
@Override
protected void onRestoreInstanceState(Bundle etat) {
    super.onRestoreInstanceState(etat);
    // restaurer l'état précédent
    mScoreJoueur = etat.getInt(ETAT_SCORE);
}
```

Ces deux méthodes sont appelées automatiquement (sorte d'écouteurs), sauf si l'utilisateur *tue* l'application. Cela permet de reprendre l'activité là où elle en était.

Vues et activités

Obtention des vues

La méthode `setContentView` charge une mise en page (*layout*) sur l'écran. Ensuite l'activité peut avoir besoin d'accéder aux vues, par exemple lire la chaîne saisie dans un texte. Pour cela, il faut obtenir l'objet Java correspondant.

```
EditText nom = findViewById(R.id.edt_nom);
```

Cette méthode cherche la vue qui possède cet identifiant dans le layout de l'activité. Si cette vue n'existe pas (mauvais identifiant, ou pas créée), la fonction retourne `null`.

Un mauvais identifiant peut être la raison d'un bug. Cela peut arriver quand on se trompe de layout pour la vue.

Propriétés des vues

La plupart des vues ont des *setters* et *getters* Java pour leurs propriétés XML. Par exemple `TextView`.

En XML :

```
<TextView android:id="@+id/titre"
    android:lines="2"
    android:text="@string/debut" />
```

En Java :

```
TextView tvTitre = findViewById(R.id.titre);
tvTitre.setLines(2);
tvTitre.setText(R.string.debut);
```

Consulter leur documentation pour les propriétés, qui sont extrêmement nombreuses.

Actions de l'utilisateur

Prenons l'exemple de ce `Button`. Lorsque l'utilisateur appuie dessus, cela déclenche un *événement* « onClick », et appelle automatiquement la méthode `Valider` de l'activité.

```
<Button
    android:id="@+id/btn_valider"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/valider"
    android:onClick="Valider" />
```

Il faut définir la méthode `Valider` dans l'activité :

```
public void Valider(View btn) {
    ...
}
```

Définition d'un écouteur

Il y a une autre manière de définir une réponse à un clic : un écouteur (*listener*). C'est une instance de classe qui possède la méthode `public void onClick(View v)` ainsi que spécifié par l'interface `View.OnClickListener`.

Cela peut être :

- une classe privée anonyme,
- une classe privée ou public dans l'activité,
- l'activité elle-même.

Dans tous les cas, on fournit cette instance en paramètre à la méthode `setOnClickListener` du bouton :

```
btn.setOnClickListener(ecouteur);
```

Écouteur privé anonyme

Il s'agit d'une classe qui est définie à la volée, lors de l'appel à `setOnClickListener`. Elle ne contient qu'une seule méthode.

```
Button btn = findViewById(R.id.btn_valider);
btn.setOnClickListener(
    new View.OnClickListener() {
        public void onClick(View btn) {
            // faire quelque chose
        }
    });
```

Dans la méthode `onClick`, il faut employer la syntaxe `MonActivity.this` pour manipuler les variables et méthodes de l'activité sous-jacente.

Écouteur privé

Cela consiste à définir une classe privée dans l'activité ; cette classe implémente l'interface `OnClickListener` ; et à en fournir une instance en tant qu'écouteur.

```
private class EcBtnValider implements OnClickListener {
    public void onClick(View btn) {
        // faire quelque chose
    }
};
public void onCreate(...) {
    ...
    Button btn = findViewById(R.id.btn_valider);
    btn.setOnClickListener(new EcBtnValider());
}
```

L'activité elle-même en tant qu'écouteur

Il suffit de mentionner `this` comme écouteur et d'indiquer qu'elle implémente l'interface `OnClickListener`.

```
public class EditActivity extends Activity
        implements OnClickListener {
    public void onCreate(...) {
        ...
        Button btn = findViewById(R.id.btn_valider);
        btn.setOnClickListener(this);
    }
    public void onClick(View btn) {
        // faire quelque chose
    }
```

Ici, par contre, tous les boutons appelleront la même méthode.

Distinction des émetteurs

Dans le cas où le même écouteur est employé pour plusieurs vues, il faut les distinguer en se basant sur leur identitifiant obtenu avec `getId()` :

```
public void onClick(View v) {
    switch (v.getId()) {
    case R.id.btn_valider:
        ...
        break;
    case R.id.btn_effacer:
        ...
        break;
    }
}
```

Événements des vues courantes

Vous devrez étudier la documentation. Voici quelques exemples :

- `Button` : `onClick` lorsqu'on appuie sur le bouton, voir sa doc
- `Spinner` : `OnItemSelected` quand on choisit un élément, voir sa doc
- `RatingBar` : `OnRatingBarChange` quand on modifie la note, voir sa doc
- etc.

Heureusement, dans le cas de formulaires, les actions sont majoritairement basées sur des boutons.