الجامعة الدولية للرباط
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵜⴰⵅⴰⵢⵜⴰⵜ ⵏ ⵕⵕⴱⴰⵟ
Université Internationale de Rabat

Web Avancé et Développement mobile

PROF. RAHHAL IBRAHIM
EMAIL: BRAHIM.RAHHAL@UIR.AC.MA

Année Universitaire: 2024-2025

Développement Mobile sous Android

Création et Gestion des activités

Introduction

L'interface utilisateur d'une application Android est composée d'écrans

→Un «écran »correspond à **une activité (Activity)** (afficher des informations, éditer des informations, …)

Android permet de naviguer d'une activité à l'autre:

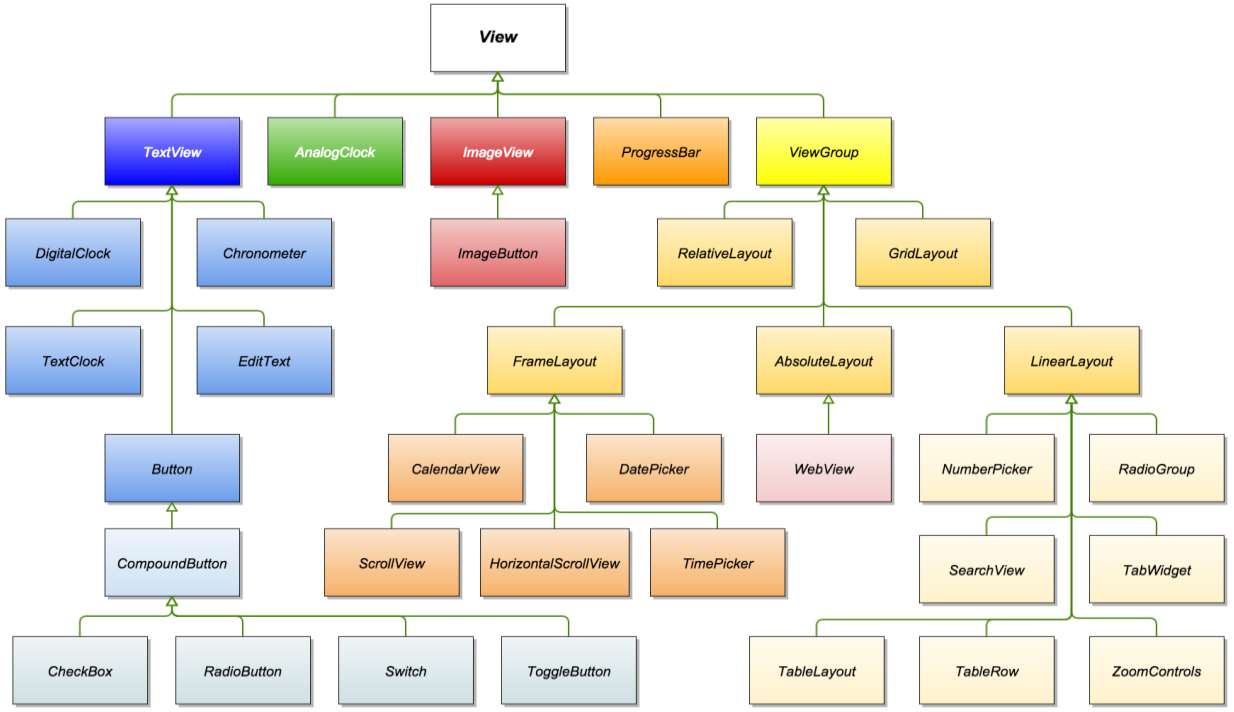
- Une action de l'utilisateur, bouton, menu
- l'application fait aller sur l'écran suivant, le bouton back ramène sur l'écran précédent.

L'interface d'une activité est composée de vues :

- Vues élémentaires : boutons, zones de texte, cases à cocher. . .
- Vues de groupement qui permettent l'alignement des autres vues : lignes, tableaux, onglets, panneaux à défilement. . .

Introduction

Chaque vue d'une interface est gérée par un objet Java : Il y a une hiérarchie de classes dont la racine est **View**. Elle a une multitude de sous-classes, dont par exemple **TextView**, elle-même ayant des sous-classes, par exemple **Button**.



Les propriétés des objets sont généralement visibles à l'écran : titre, taille, position, etc.

Introduction :

Chaque écran est géré par une instance d'une sous-classe de **Activity** que vous programmez. Il faut au moins surcharger la méthode **onCreate** selon ce qui doit être affiché sur l'écran.

Bundle : une classe permettant d'offrir un conteneur pour les données transmissibles d'une activité à l'autre.

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    }
}
```

Chaque activité a son fichier de mise en page sous forme xml (**MainActivity.xml**) : nous incluons sa mise en page à l'aide de la méthode **setContentView** de la classe Activity à l'aide de l'identifiant de ressource : **R.layout.activity_main**

C'est quoi la classe R ?

Les Ressources

Les ressources sont tout ce qui n'est pas programmé dans une application (les textes, messages, icones, images, sons, interfaces, styles, etc.)

→ permettent d'adapter une application facilement à tous les spécificités (les pays, cultures et langues) : pas besoin de recompiler le code source à chaque fois mais juste choisir des ressources spécifiques.

Exemple : Pour avoir une application multilingue on pout prévoir des variantes linguistiques associées à des ressources qu'on veut traduire : créer des sous-dossier par exemple ex: values-ar, values-en, values-fr, etc et il n'y a qu'à modifier des fichiers XML.

Le problème est alors de faire le lien entre les ressources et les programmes :

→ Utilisation d'un identifiant : un entier qui est généré automatiquement par le SDK Android. Les identifiants ont été catégorisés par type (par exemple chaque vue, image, icone texte, message etc… possède un identifiant)

Ils ont tous été regroupés dans une classe spéciale appelée R.

Les Ressources : la classe R

Le SDK Android, à l'aide de l'AAPT « Android Asset Packaging Tool », prend les fichiers de ressource d'une application, tels que le fichier AndroidManifest.xml et les fichiers XML des «Activity »et les compile.

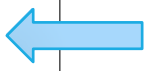
→ Une classe R est générée automatiquement (dans le dossier generated) par ce que vous mettez dans le dossier res : interfaces, menus, images, chaînes. . . Certaines de ces ressources sont des fichiers XML, d'autres sont des images PNG.

R.java est produit afin de permettre de référencer les ressources depuis le code Java.

```
public final class R {
    public static final class string {
        public static final int app_name = 0x4f060000;
        public static final int message = 0x4f060001;
    }

    public static final class layout {
        public static final int main = 0x4f030000;
    }

    public static final class menu {
        public static final int main_menu = 0x4f050000;
        public static final int context_menu = 0x4f050001;
    }
```



strings.xml

Edit translations for all locales in the translations editor.

```
1  <resources>
2      <string name="app_name">My Application</string>
3      <string name="message">bonjour</string>
4  </resources>
5
6
```

Les Ressources de type chaînes et Traduction

Dans res/values/strings.xml, on place les chaînes de l'application, au lieu de les mettre en constantes dans le source :

```
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="main_menu">Menu principal</string>
    <string name="action_settings">Configuration</string>
    <string name="bonjour">Bonjour !</string>
</resources>
```

Lorsque les textes sont définis dans res/values/strings.xml, il suffit de faire des copies du dossier values, en values-us, values-ar, etc. et de traduire les textes en gardant les attributs name. Voici par exemple res/values-us/strings.xml :

```
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="main_menu">Main Menu</string>
    <string name="action_settings">Settings</string>
    <string name="bonjour">Hello !</string>
</resources>
```

Le système android ira chercher automatiquement le bon texte en fonction des paramètres linguistiques configurés par l'utilisateur.

Référencement des ressources:

Dans un programme Java, on peut placer un texte dans une vue de l'interface :

```
TextView param_textVw = new TextView( context: this);
param_textVw.setText(R.string.bonjour);
```

R.string.bonjour désigne le texte de <string name="bonjour">... dans le fichier res/values*/strings.xml

TextView.setText() a deux versions :

- void setText(String text) : on peut fournir une chaîne quelconque
- void setText(int idText) : on doit fournir un identifiant de ressource chaîne, donc forcément l'un des textes du fichier res/values/strings.xml

Pour récupérer l'une des chaînes des ressources pour l'utiliser dans le programme :

```
String bonj_msg = getResources().getString(R.string.bonjour);
```

getResources() est une méthode de la classe Activity (héritée de la classe abstraite Context) qui retourne une représentation de toutes les ressources du dossier res. Chacune de ces ressources, selon son type, peut être récupérée avec son identifiant.

Référencement des ressources

Dans un fichier de ressources res/values/arrays.xml décrivant un tableau des chaines R.array.nom :

```xml
<resources>
    <string-array name="Joursdutravail">
        <item>Lundi</item>
        <item>Mardi</item>
        <item>Mercredi</item>
        <item>Jeudi</item>
        <item>Vendredi</item>
    </string-array>
</resources>
```

Dans le programme Java, il est possible de faire :

```java
Resources resources = getResources();
String[] jours = resources.getStringArray(R.array.Joursdutravail);
```

Référencement des ressources dans une interface

Dans un fichier de ressources décrivant une interface, on peut employer des ressources texte :

```
<RelativeLayout>
    <TextView android:text="@string/bonjour" />
</RelativeLayout>
```

@string/nom est une référence à la chaîne du fichier res/values*/strings.xml ayant ce nom.

Les images PNG placées dans res/drawable et res/mipmaps-* sont référençables :

```
<ImageView
    android:src="@drawable/img"
    android:contentDescription="@string/mon_img" />
```

La notation @drawable/nom référence l'image portant ce nom dans l'un des dossiers.

Les dossiers res/mipmaps-* contiennent la même image à des définitions différentes, pour correspondre à différents smartphones et tablettes. Par exemple : mipmap-hdpi contient des icônes en 72x72 pixels.

Autres notations

D’autres notations existent :

- @style/nom pour des définitions de res/style
- @menu/nom pour des définitions de res/menu

Certaines notations, @package:type/nom font référence à des données prédéfinies, comme :

- @android:style/TextAppearance.Large
- @android:color/black

Il y a aussi une notation en ?type/nom pour référencer la valeur de l’attribut nom, ex : ?android:attr/textColorSecondary.

Les interfaces

Un écran Android de type formulaire est généralement composé de plusieurs vues :

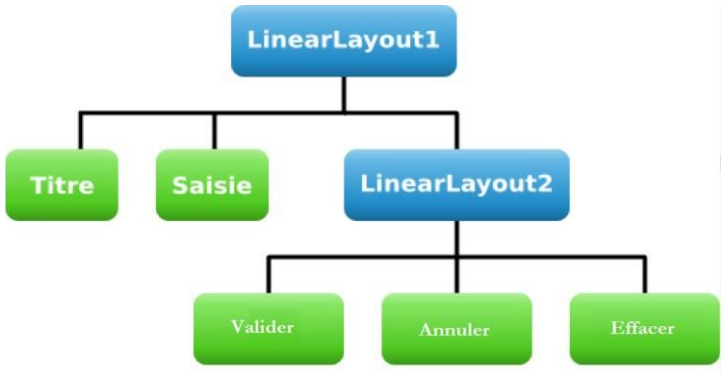
- TextView, ImageView : titre, image
- EditText : texte à saisir
- Button, CheckBox : bouton à cliquer, case à cocher

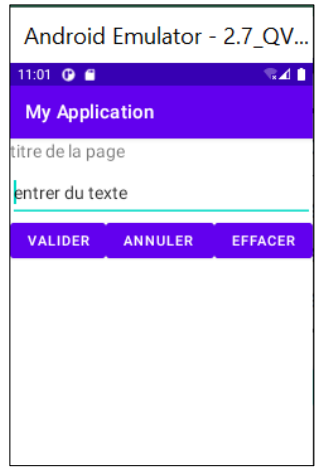
Ces vues sont alignées à l'aide de groupes sous-classes de ViewGroup, éventuellement imbriqués :

- LinearLayout : positionne ses vues en ligne ou en colonne
- RelativeLayout, ConstraintLayout : positionnent leurs vues l'une par rapport à l'autre
- TableLayout : positionne ses vues sous forme d'un tableau

Les groupes et vues forment un arbre :

Arbre des vues





Les interfaces : Création par programme

Il est possible de créer une interface par programme :

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    TextView ecriture = new TextView( context: this);
    ecriture.setText(R.string.bonjour);
    LinearLayout ressl = new LinearLayout( context: this);
    LayoutParams lp = new LayoutParams();
    lp.width = LinearLayout.LayoutParams.MATCH_PARENT;
    lp.height = LinearLayout.LayoutParams.MATCH_PARENT;
    ressl.addView(ecriture, lp);
    setContentView(ressl);



}
```

Les interfaces : Ressources de type layout

Il est donc préférable de stocker l'interface dans un fichier res/layout/nom_du_fichier.xml (exemple res/layout/main.xml) :

```
<LinearLayout >
    <TextView
        android:id="@+id/textView3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="titre de la page"
</LinearLayout>
```

qui est référencé par son identifiant R.layout.nom_du_fichier (par exemple R.layout.main) dans le programme Java et la méthode setContentView permet d'afficher le layout indiqué.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
```

Les interfaces : Identifiants et vues

Lorsque l'application veut manipuler l'une de ses vues, elle doit utiliser R.id.symbole, exemple :

```
TextView textview = findViewById(R.id.message);
```

Dans les fichiers layouts, il y a deux notations à ne pas confondre :

- @+id/nom pour définir (créer) un identifiant
- @id/nom pour référencer un identifiant déjà défini ailleurs

Exemple, le Button btn se place sous le TextView titre :

```
<RelativeLayout xmlns:android="..." >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/titre"
        android:text="@string/titre" />
    <Button
        android:id="@+id/btn"
        android:layout_below="@id/titre"
        android:text="@string/ok" />
</RelativeLayout>
```

Les interfaces : Paramètres de positionnement

La plupart des groupes utilisent des paramètres de taille et de placement sous forme d'attributs XML. Ces paramètres sont de deux sortes :

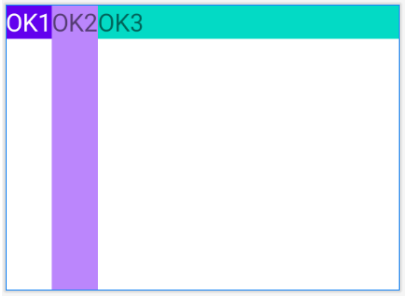
- Obligatoires pour toutes les vues :
  - android:layout_width : largeur de la vue
  - android:layout_height : hauteur de la vue

  Ils peuvent valoir :
  - "wrap_content" : la vue prend la place minimale
  - "match_parent" : la vue occupe tout l'espace restant
  - "valeurdp" : une taille fixe, ex : «100dp »(dp : unité de taille indépendante de l'écran)
- Sont demandés par le groupe englobant et qui en sont spécifiques, comme android:layout_weight (spécifier un rapport de taille entre plusieurs vues), android:layout_alignParentBottom, android:layout_centerInParent.

Les interfaces : Paramètres de positionnement

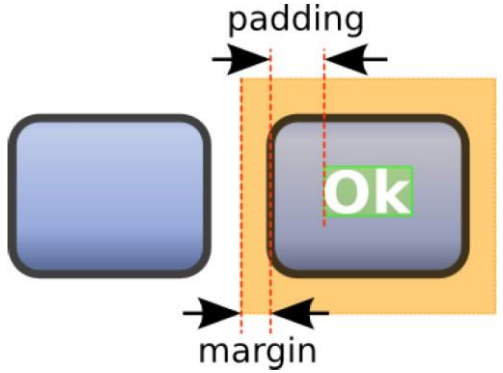
Par exemple, trois boutons dans un LinearLayout horizontal :

Bouton	layout_width	layout_height
OK1	wrap_content	wrap_content
OK2	wrap_content	match_parent
OK3	match_parent	wrap_content



Il est possible de modifier l'espacement des vues :

- Padding espace entre le texte et les bords, géré par chaque vue
- Margin espace autour des bords, géré par les groupes



On peut définir les marges et les remplissages séparément sur chaque bord (Top, Bottom, Left, Right), ou identiquement sur tous

```
<Button
android:layout_margin="10dp"
android:layout_marginTop="15dp"
android:padding="10dp"
android:paddingLeft="20dp" />
```

Les interfaces : Groupe de vues LinearLayout

Il range ses vues soit horizontalement, soit verticalement: Il faut seulement définir l'attribut android:orientation à "horizontal" ou "vertical".

```
<LinearLayout android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <Button android:text="Ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <Button android:text="Annuler"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

Les interfaces : Groupe de vues LinearLayout/ TableLayout

LinearLayout: range ses vues soit horizontalement, soit verticalement: Il faut seulement définir l'attribut android:orientation à "horizontal" ou "vertical".

```
<LinearLayout android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <Button android:text="Ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <Button android:text="Annuler"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

TableLayout : C'est une variante du LinearLayout : les vues sont rangées en lignes de colonnes bien alignées. Il faut construire une structure XML comme celle-ci. (les <TableRow> n'ont aucun attribut).

```
<TableLayout ...>
    <TableRow>
        <vue 1.1 .../>
        <vue 1.2 .../>
    </TableRow>
    <TableRow>
        <vue 2.1 .../>
        <vue 2.2 .../>
    </TableRow>
<TableLayout>
```

Les vues (android.widget)

Android propose un grand nombre de vues :

- Textes : titres, chaînes à saisir
- Boutons, cases à cocher. . .
- Curseurs : pourcentages, barres de défilement…

Elle se présente avec de variantes. Exemple: saisie de texte = no de téléphone, ou adresse, ou texte avec suggestion, ou . . .

TextView : Le plus simple, il affiche un texte statique, comme un titre. Son libellé est dans l'attribut android:text.

```
<TextView
    android:id="@+id/tvtitre"
    android:text="@string/titre"
    ... />
```

```
TextView tvTitre = findViewById(R.id.tvtitre);
tvTitre.setText("dunouvtexte");
```

Les vues (android.widget)

Button : L'une des vues les plus utiles:

- on définit un identifiant pour chaque vue active, ici : android:id="@+id/btn_ok"
- Son titre est dans l'attribut android:text.
- réaction à un clic (on va la voir après)

```
<Button
    android:id="@+id/btn_ok"
    android:text="@string/ok"
... />
```

CheckBox : sont des cases à cocher :

Les **ToggleButton** sont une variante : . On peut définir le texte actif/inactif avec android:textOn et android:textOff

```
<CheckBox
    android:id="@+id/cbx_abonnement_nl"
    android:text="@string/abonnement_newsletter"
... />
```

EditText: permet de saisir un texte: L'attribut android:inputType spécifie le type de texte : adresse, téléphone, etc. Ça définit le clavier qui est proposé pour la saisie.

```
<EditText
    android:id="@+id/email_address"
    android:inputType="textEmailAddress"
... />
```

Applications et activités

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.

```
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ...>
        <activity android:name=".MainActivity" ... />
        <activity android:name=".ConfigActivity" ... />
        <activity android:name=".DessinActivity" ... />
        ...
    </application>
</manifest>
```

Le fichier AndroidManifest.xml déclare les éléments d'une application, avec un '.' devant le nom de classe des activités : Une activité qui n'est pas déclarée dans le manifeste ne peut pas être lancée.

Démarrage d'une application / d'une activité

L'une des activités est marquée comme étant démarrable de l'extérieur, grâce à un sous-élément <intent-filter> :

```
<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. Celui-ci indique l'activité principale, celle qu'il faut lancer quand on clique sur son icône.

Les Intents permettent de gérer l'envoi et la réception de messages afin de faire coopérer les applications. Le but des Intents est de déléguer une action à un autre composant, une autre application ou une autre activité de l'application courante.

Démarrage d’une application / d’une activité

la déclaration d'intention (intent) de type "media" est utilisée pour déclarer les capacités multimédias de votre application. Elle informe le système Android que votre application peut effectuer des opérations liées aux médias (lecture, enregistrement)

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

L'ajout de cette déclaration dans le fichier manifest de votre application permet d'indiquer aux autres applications ou aux composants du système Android que votre application peut gérer des opérations multimédias. Par exemple, si une autre application souhaite lancer un lecteur audio pour jouer un fichier audio, elle peut rechercher les applications qui déclarent la capacité de gérer les médias, et votre application sera incluse dans la liste des choix possibles.

Démarrage d'une application / d'une activité

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.

Soit une application contenant deux activités Activ1 et Activ2.:

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

- La première lance la seconde par L'instruction startActivity ==> démarre Activ2. Celle-ci se met au premier plan, tandis que Activ1 se met en sommeil:
- Activ1 reviendra au premier plan quand Activ2 se finira ou quand l'utilisateur appuiera sur back.

Autorisations et sécurité 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 éléments dans le manifeste :

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

</manifest>
```

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

Chaque application est associée à un UID (compte utilisateur Unix) unique dans le système. Ce compte les protège les unes des autres. Il peut être défini dans le fichier AndroidManifest.xml sous forme d'un nom de package :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    android:sharedUserId="ma.uir.demos">
    ...
</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.

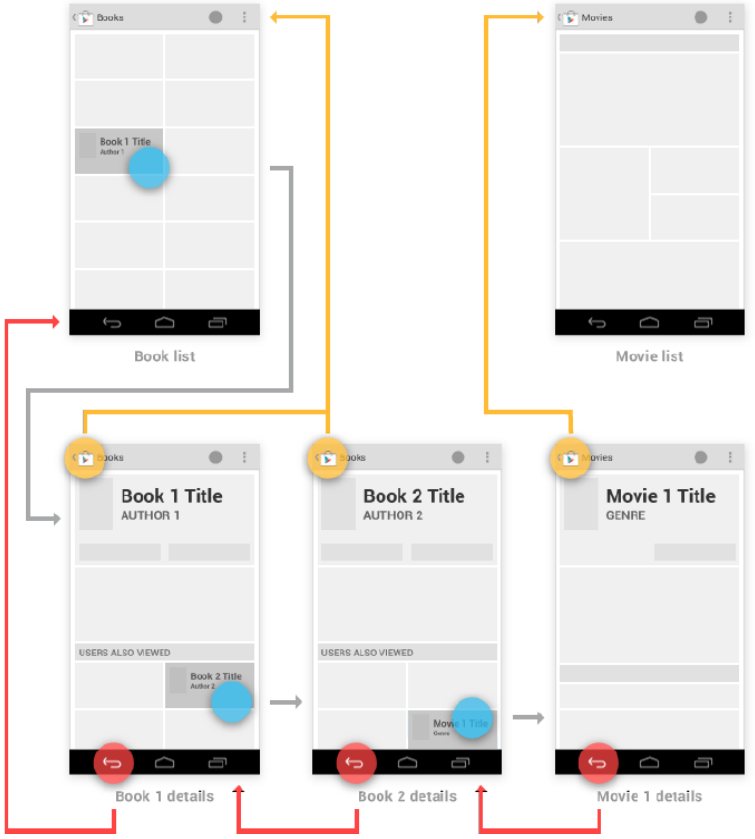
Les Applications

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. Par 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.



Les Applications : Lancement des activités

Lancement d'une activité avec retour

```
//pour lancer Activ2 à partir de Activ1 :
Intent intent = new Intent( packageContext: this, Activ2.class);
startActivity(intent);
```

Lancement d'une activité sans retour

```
//On peut demander la terminaison de cette activité après lancement de Activ2  :
Intent intent = new Intent( packageContext: this, Activ2.class);
startActivity(intent);
finish();
```

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

Les Applications : Transport/Extraction d'informations

Les Intents servent aussi à transporter des informations d'une activité à l'autre à l'aide **des extras.**

```
Intent intent = new Intent( packageContext: this, DeleteProprieteActivity.class);
intent.putExtra( name: "idPropriete", idProp);
intent.putExtra( name: "proprieteAssociee", propAsc);
startActivity(intent);
```

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

Ces instructions récupèrent les données d'un Intent :

```
Intent intent = getIntent();
Integer idProp = intent.getIntExtra( name: "idPropriete",  defaultValue: -1);
Boolean propAsc = intent.getBooleanExtra( name: "proprieteAssociee",  defaultValue: false);
```

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

Les Applications : le contexte de l'application

Il existe un objet global vivant pendant tout le fonctionnement d'une application : le contexte d'application. On peut le récupérer :

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

Par défaut, c'est un objet neutre ne contenant que des informations Android: Il est possible des sous-classes permettant de stocker des variables globales de l'application.

```
public class MonAppli extends Application
{
    // definition d'un variable globale de l'application
    private int globvar;
    public int getGlobVar() {
        return globvar;
    }

    // initialisation du Contexte
    @Override
    public void onCreate() {
        super.onCreate();
        globvar = 5;
    }
}
```

Les Applications : le contexte de l'application

Après, il faut déclarer dans AndroidManifest.xml, l'attribut **android:name** de l'élément <application>, en mettant un point devant :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapplication">

    <application android:name=".MonAppli"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
```

Et on peut l'utiliser dans n'importe laquelle des activités :

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

// utiliser la variable globale
int nombre = context.getGlobVar();
```

Les Activités

Une seule activité visible à l'écran au premier plan.

Une activité peut se trouver dans 3 états qui se différencient surtout par leur visibilité :

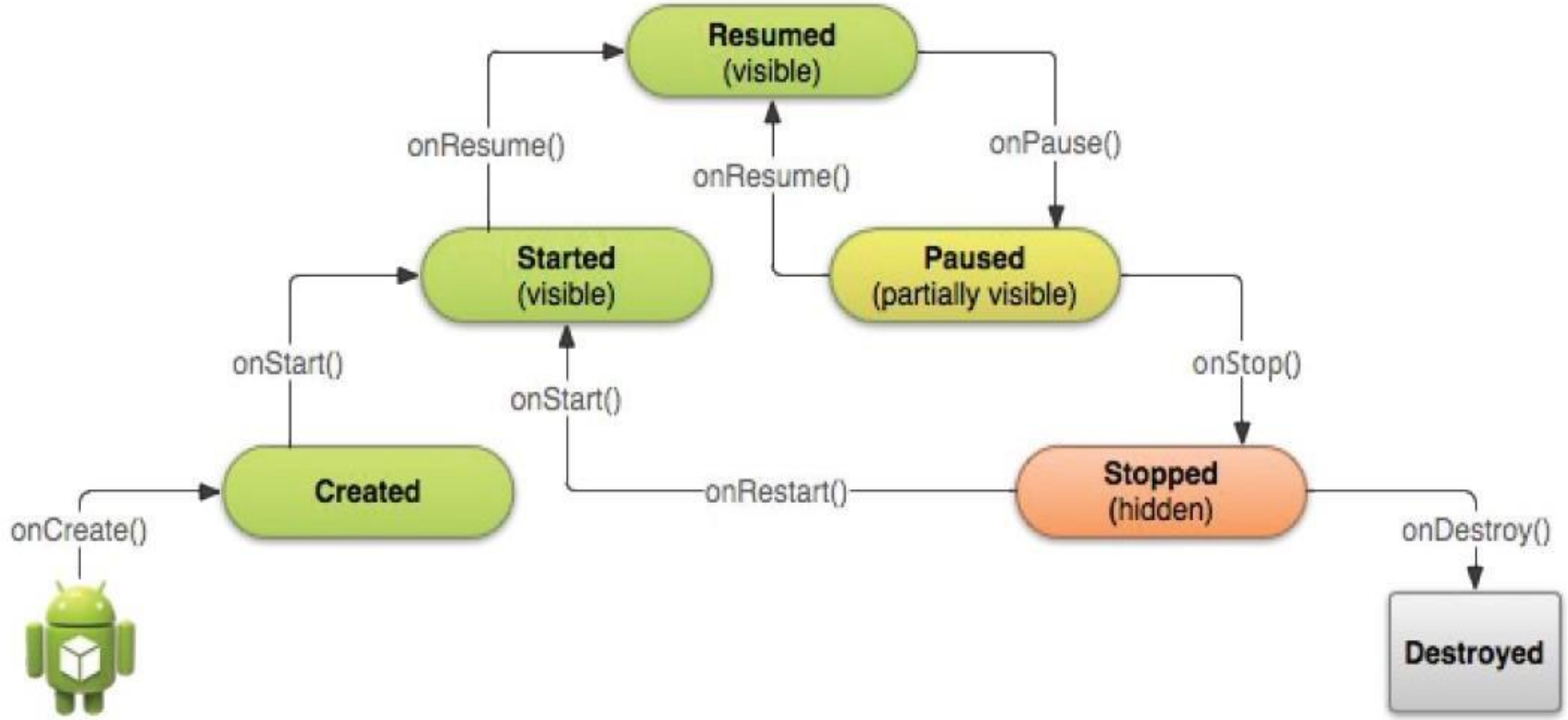
- **Active (started / resumed)** : elle est sur le devant, l'utilisateur peut jouer avec.
- **Suspendue (paused)** : partiellement cachée et inactive, car une autre activité est venue devant.
- **Arrêtée (stopped)** : totalement invisible et inactive, ses variables sont préservées mais elle ne tourne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

Passage de l'état visible à invisible :

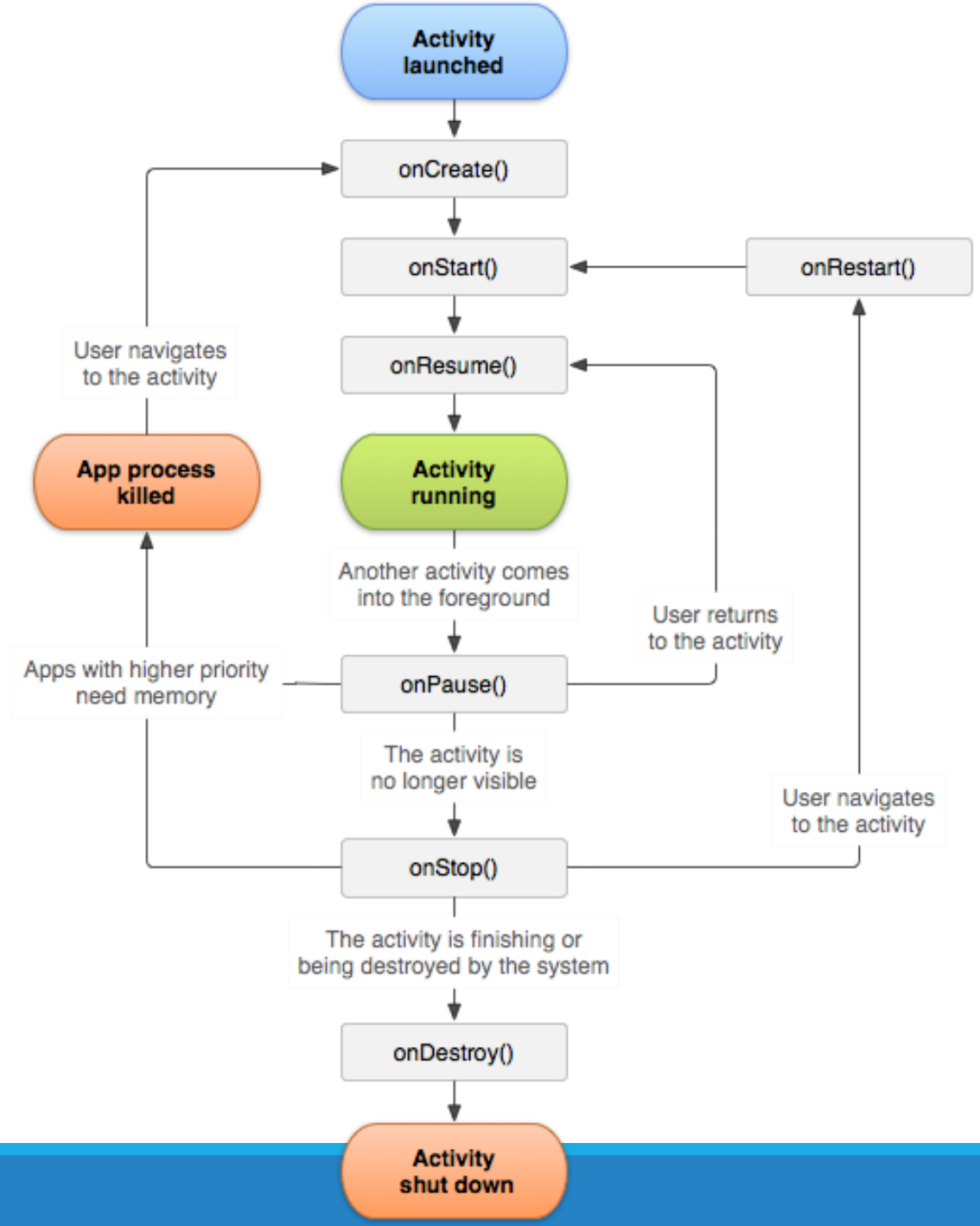
- En lançant une nouvelle activité masquant l'activité courante
- En détruisant l'activité courante (finish(), bouton "retour");

Les Activités : Cycle de vie

L'activité réagie à la demande du système Android (qui lui même réagi à la demande de l'utilisateur):



Cycle de vie d'une activité



Les Activités : Activity.onXXX()

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

- onCreate(Bundle savedInstanceState) : appelée à la création. Le paramètre permet de récupérer un état sauvegardé lors de l'arrêt de l'activité.
- onStart() : appelée quand l'activité démarre
- onRestart() : appelée quand l'activité redémarre
- onResume() : appelée quand l'activité vient en premier plan
- onPause() : appelée quand l'activité n'est plus en premier plan
- onStop() : appelée quand l'activité n'est plus visible
- onDestroy() : appelée quand l'activité se termine

On redéfinie les méthodes que l'on veut et on n’oublie pas d'appeler en premier super.onXXX()

Les Activités : Squelette d'activité et méthodes

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

@Override signifie que cette méthode remplace celle héritée de la superclasse.

```
public class EcritureActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState) {
        // obligatoire
        super.onCreate(savedInstanceState);
        // met en place les vues de cette activité
        setContentView(R.layout.ecriture_activity);
    }
}
```

Utiliser **onDestroy** : la prise en compte de la terminaison définitive d'une activité en spécifiant des actions a faire avant (tels que la fermeture d'une base de données).

```
@Override
public void onDestroy() {
    // obligatoire
    super.onDestroy();
    // fermer la base de données
    basedonnees.close();
}
```

Les Activités : Squelette d'activité et méthodes

Utiliser **onPause**: Le système prévient l'activité qu'une autre activité est passée devant : il faut enregistrer les informations et libérer les ressources qui consomment de l'énergie.

Utiliser **onResume** : Le système prévient que l'activité revient en premier plan: il faut récupérer les informations et les ressources utilisées par cette activité.

Exemple: Lors d'un appel téléphonique

```
@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 : 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. On se dispose de deux méthodes :

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

```
@Override
protected void onStop()  {
    super.onStop();
    // libérer les ressources,
    ....
}
@Override
protected void onStart() {
    super.onStart();
// allouer les ressources.
    ....
}
```

Les Activités : Enregistrer et Restaurer l'état

Il est possible de sauver des informations d'un lancement à l'autre de l'application dans un **Bundle**: un container de données quelconques, sous forme de couples **("nom", valeur).**

```
static final String Nom_Etudiant = "Etudiant1"; // nom
private int ordreetudiant = 5; // valeur
@Override
public void onSaveInstanceState(Bundle etat) {
    // enregistrer l'état courant
    etat.putInt(Nom_Etudiant, ordreetudiant);
    super.onSaveInstanceState(etat);
}
```

La méthode onRestoreInstanceState reçoit un paramètre de type Bundle (comme 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
    ordreetudiant = etat.getInt(Nom_Etudiant);
}
```

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 : ViewBindings

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): il faut obtenir l'objet Java:

```
EditText nom = findViewById(R.id.edit_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 et ca peut être la raison d'un bug.

Pour éviter les problèmes de typage et de vues absentes d'un layout, il existe un dispositif appelé **ViewBindings:** des classes qui sont générées automatiquement à partir de chaque layout et dont les variables membres sont les différentes vues.

Vues et activités: ViewBindings

```
<LinearLayout xmlns:tools="http://schemas.android.com/tools"
    <TextView android:id="@+id/titre" .../>
    <Button android:id="@+id/buttonOk" .../>
</LinearLayout>
```

Permet de générer une classe appelée ActivityMainBinding.java (à peu près ceci):

```
public final class ActivityMainBinding implements ViewBinding {
    private final LinearLavout rootView; // voir getRoot()
    public final Button btnOk;
    public final TextView titre;
    ...
}
```

Chaque vue du layout xml possédant un identifiant est reliée à une variable membre publique dans cette classe, et la vue racine est accessible par **getRoot**().

Une méthode statique **inflate** instancie les différentes vues et la vue racine peut être fournie à setContentView.

Vues et activités: ViewBindings

Dans une activité, on peut faire ceci :

```
public final class MainActivity extends Activity
{
    // ui = interface utilisateur.
    private MainActivity.ActivityMainBinding userinter;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        userinter = MainActivity.ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(userinter.getRoot());
        // exemple d'emploi
        userinter.titre.setText("super cool !");
    }
```

Et pour générer les ViewBindings Il faut rajouter ceci dans app/build.gradle

```
android {
    compileSdkVersion "..."
    defaultConfig {
        ....
    }
    buildFeatures {
        viewBinding = true // génération des ViewBindings
    }
```

Vues et activités : Actions de l'utilisateur

Lorsque l'utilisateur appuie sur un bouton, ça appelle automatiquement la méthode spécifique (dans ce cas onValider) de l'activité grâce à l'attribut onClick="onValider".

```
<Button
    android:onClick="onValider"
    android:id="@+id/btnValider"
    android:layout_width="wrap_content"
```

```
//Il faut définir la méthode onValider dans l'activité
public void onValider(View btn) {
    ....
}
```

Pour définir une réponse à un clic : on utilise un **écouteur** (**listener**) : une instance de classe (classe privée anonyme, classe privée ou publique dans l'activité, ou l'activité elle-même) implémentant l'interface **View.OnClickListener** qui possède la méthode **public void onClick(View v).**

On fournit cette instance en paramètre à la méthode setOnClickListener du bouton :

```
Button button = userinter.buttonvalider;
button.setOnClickListener(ecouteur);
```

Vues et activités : Écouteur privé anonyme

C'est une classe qui est définie à la volée, lors de l'appel à setOnClickListener. Elle ne contient qu'une seule méthode : Dans la méthode onClick, il faut employer la syntaxe MonActivity.this pour manipuler les variables et méthodes de l'activité sous-jacente.

```
Button button = userinter.buttonvalider;
button.setOnClickListener(
        new View.OnClickListener() {
            public void onClick(View button) {
        // faire quelque chose
            }
        });
```

Il est intéressant de transformer cet écouteur en expression lambda.

```
Button button = userinter.buttonvalider;
button.setOnClickListener((View button) -> {
// faire quelque chose
});
```

Vues et activités : 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 View.OnClickListener {
    public void onClick(View btn) {
    // faire quelque chose
    }
};
    public void onCreate(...) {
    ...
    Button btn = userinter.buttonvalider;
    btn.setOnClickListener(new EcBtnValider());
}
```

Vues et activités : L'activité elle-même en tant qu'écouteur

Il suffit de mentionner this comme écouteur et d'indiquer qu'elle implémente l'interface OnClickListener: dans ce cas tous les boutons appelleront la même méthode.

```
public class EditActivity extends Activity implements View.OnClickListener {
    public void onCreate(...) {

    Button button = userinter.buttonvalider;
    button.setOnClickListener(this);
                        }
    public void onClick(View btn) {
    // faire quelque chose
    }
```

Vues et activités : 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;
    }
```

Vues et activités : Écouteur référence de méthode

Une manière très pratique pour associer un écouteur avec une référence de méthode : son nom est précédé par l'objet qui la définit :

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Button button = userinter.button;
    button.setOnClickListener(this::onButtonClicked);
    ...
}

private void onButtonClicked(View button) {
    // executer un traitement
}
```

Événements des vues courantes :

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

QCM

https://forms.office.com/r/c7ymiGiSWp?origin=lprLink