Introduction à la programmation sous Android

Christophe Renaud

M2 Informatique

Année 2023-2024

Version 6.5.0 du 05/09/2023

Objectifs du cours

- Connaître les bases de la programmation sous Android
  - Environnement de développement (Android Studio)
  - Architecture d'une application
  - Modèle d'exécution

Plan du cours

- Introduction
- Architecture d'une application Android
- Les activités
- Définir une interface graphique
- Les intentions explicites
- Les intentions implicites
- Les menus
- Les listes
- Les permissions
- Les content providers

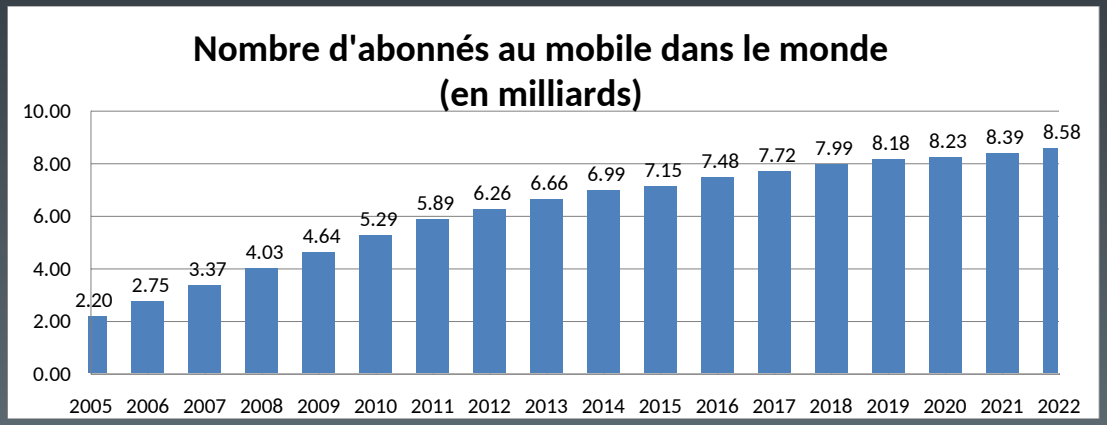
Développement mobile (1)

- Pourquoi développer des applications mobiles ?

Device Type	2021 Shipments	2021 Growth (%)	2022 Shipments	2022 Growth (%)		2023 [a] Shipments	2023 [a] Growth (%)
PC	342	11.0	287	-16.0	-9.5 [b]	267	-6.8
Tablet	156	-0.8	136	-12.0	-9 [b]	132	-2.9
Mobile Phone	1,567	5.0	1,395	-11.0	-7.1 [b]	1,339	-4.0
Total Devices	**2,065**	**5.5**	**1,819**	**-11.9**	**-7.6**	**1,740**	**-4.4**

(a) prévision - (b) prévu en juin 2022

Ventes en centaines de millions d'unités - source : (janvier 2023)
https://www.gartner.com/en/newsroom/press-releases/2023-01-31-gartner-forecasts-worldwide-device-shipments-to-decline-four-percent-in-2023



https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx

Développement mobile (2)

- Quelles solutions pour le développement ?

Natif	Hybride	Web
• IOS (objective C / swift) • Android (java, kotlin)	• React Native (javascript) • Ionic react (js, html, css)	• Vue.js
+ bien adapté à l'OS ciblé + ergonomie/design optimal + notification push + autonomie réseau - coût de développement (multiplateforme)	+ Temps de développement réduit + Maintenance facilitée - Performances moins stables	+ rien à installer + dvt rapide et portable - fonctionnement sous-optimal - besoin d'une connexion internet - pas de notifications push

Flutter (langage Dart)

Android (1)

- Pourquoi développer sous Android ?

Répartition des ventes de smartphones selon leur OS

Year	2018	2019	2020	2021	2022	2023	2024
Android	85.1%	86.1%	85.4%	86.0%	86.2%	86.3%	86.4%
iOS	14.9%	13.9%	14.6%	14.0%	13.8%	13.7%	13.6%
Others	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
TOTAL	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**

Source (juin 2020) : https://www.idc.com/promo/smartphone-market-share/os

Android (2)

- Système d'exploitation à destination des dispositifs mobiles
  - Téléphones, tablettes, téléviseurs, montres, voitures, objets connectés
- Caractéristiques :
  - Opensource (licence Apache), gratuit, flexible
  - Basé sur un noyau linux
  - Inclut les applications de base (téléphone, sms, carnet d'adresse, navigateur, etc.)
  - Un ensemble important d'API (OpenGL, media, etc …)
  - Un SDK basé sur un sous-ensemble de JAVA (autres langages disponibles : Kotlin, C, C++, …)
  - Une machine virtuelle (Dalvik) qui exécute la majorité des applications
    - Remplacée par ART depuis la version 5.0 d'Android

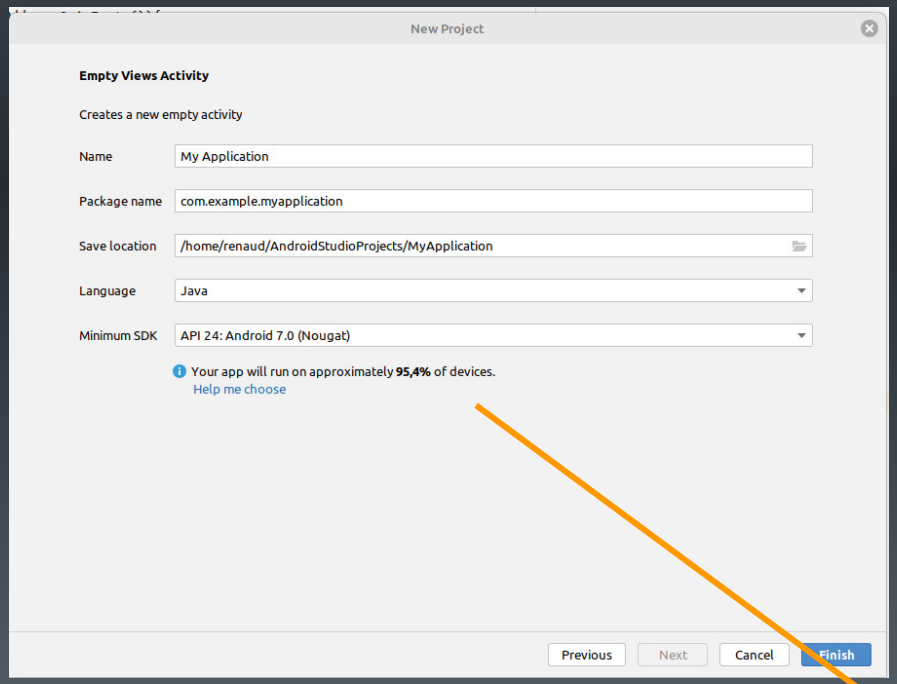
Android (3)

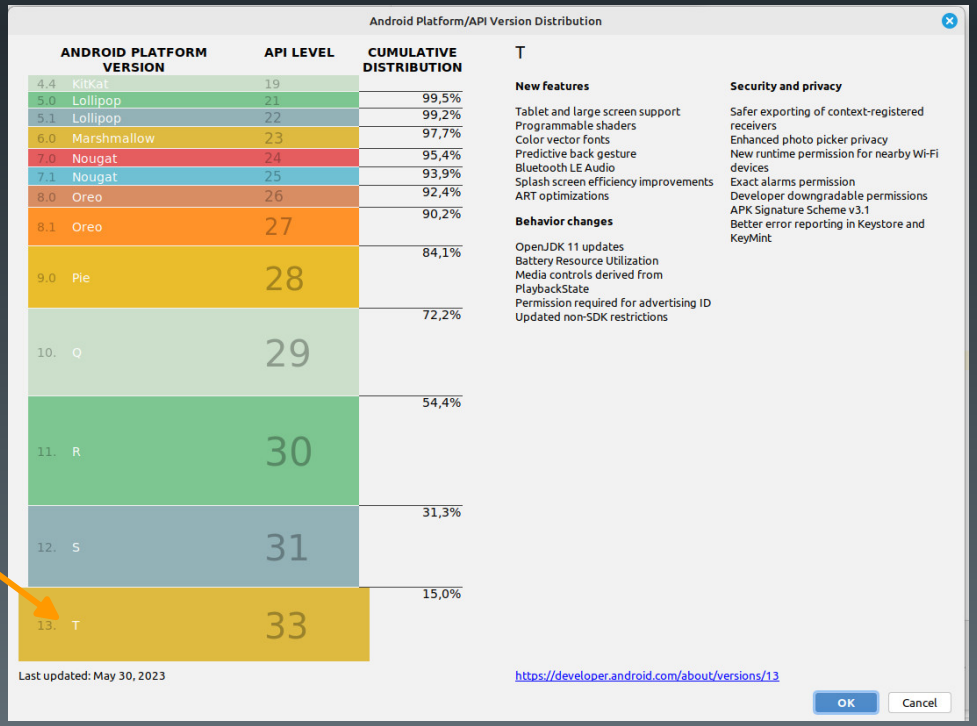
- Historique :
  - Créé en 2005 par la société Android
  - Rachat en 2007 par Google
  - 20 versions depuis la 1.0 (Apple Pie) en 2008 jusqu'à la 13.0 en 08/2022.
  - Une version = 1 API
    - Apparition de nouvelles fonctionnalités
    - Modification de fonctionnalités existantes
    - Disparition de certaines fonctionnalités

version	Nom de code	API
4.0	Ice Cream Sandwich	15
4.1	Jelly Bean	16
4.2	Jelly Bean	17
4.3	Jelly Bean	18
4.4	Kitkat	19
5.0	Lollipop	21
5.1	Lollipop	22
6.0	Marshmallow	23
7.0	Nougat	24
7.1	Nougat	25
8.0	Oreo	26
8.1	Oreo	27
9.0	Pie	28
10.0	Q (Quince Tart)	29
11.0	R (Red Velvet Cake)	30
12.0	S (Snow Cone)	31-32
13.0	T (Tiramisu)	33

Android (4)

- Comment assurer la compatibilité ?
  - Vérifier depuis Android Studio lors de la création d'un nouveau projet





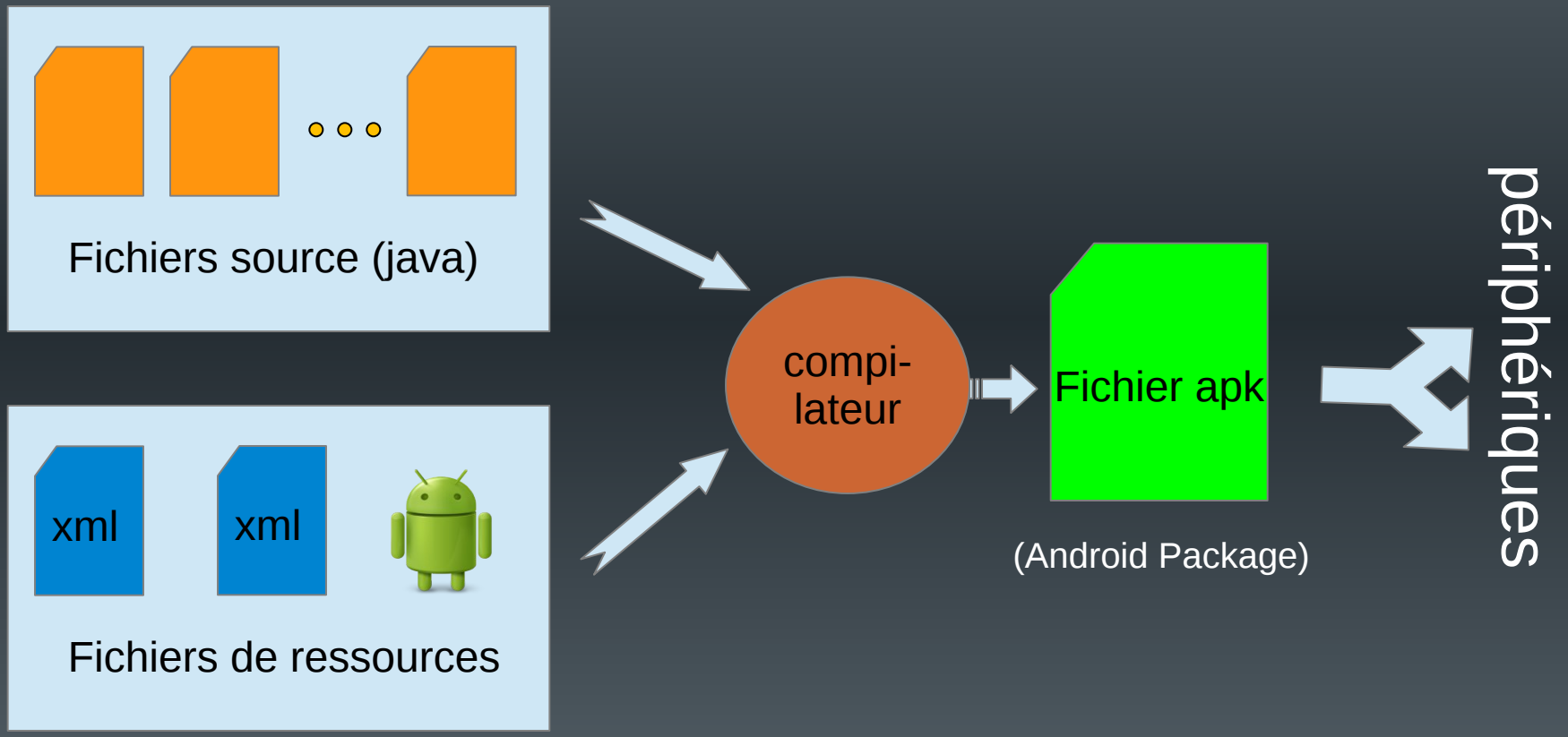
Android (5)

- Les contraintes
  - Hétérogénéité du matériel
    - Processeurs, mémoire
    - Écrans
    - Dispositifs spécialisés
  - Puissance et mémoire limitées
  - Interface tactile
  - Connectivité à internet (disponibilité, rapidité, ...)
  - Développement extérieur au périphérique

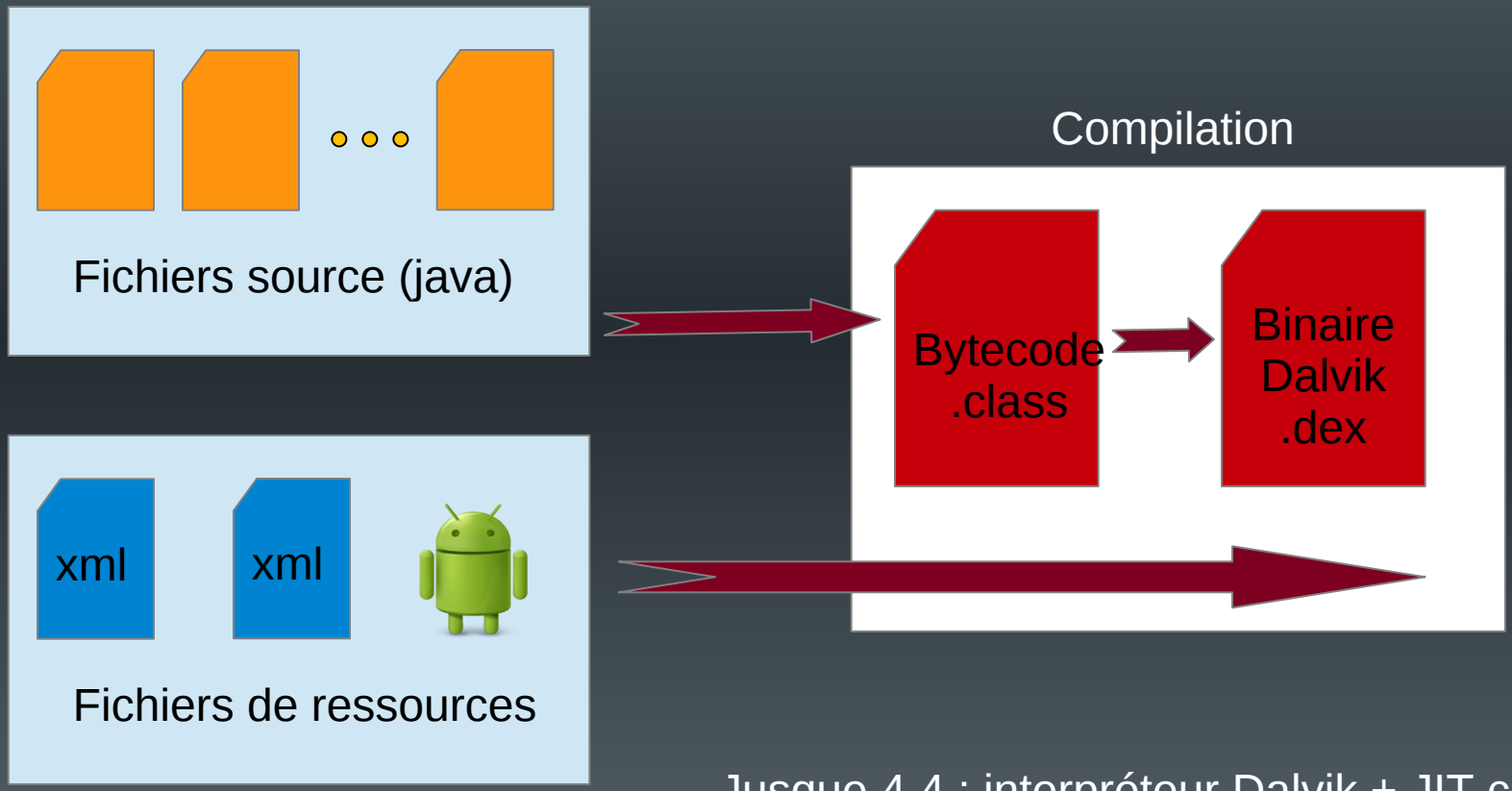
Plan du cours

- Introduction
- Architecture d'une application Android
- Les activités
- Définir une interface graphique
- Les intentions explicites
- Les intention implicites
- Les permissions
- Les menus
- Les listes
- Les content providers

Schéma de développement

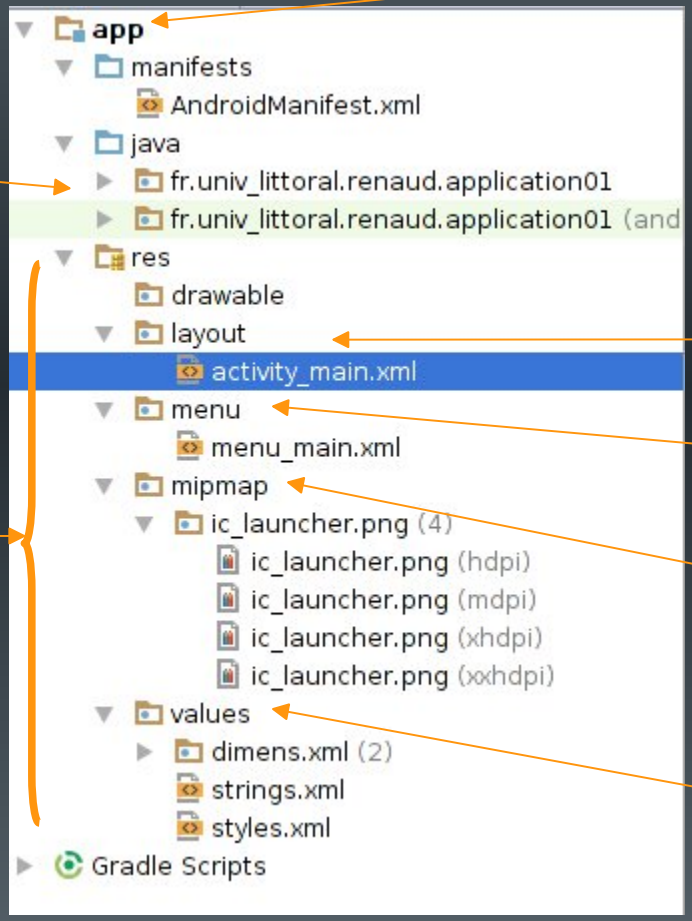


La compilation



Jusque 4.4 : interpréteur Dalvik + JIT compilation de parties « critiques »
À partir de 5.0 : ART (compilation en code natif sur le support)

Architecture d'un projet



Dossier du projet

Sources Java

Mises en page

Menus

Les ressources

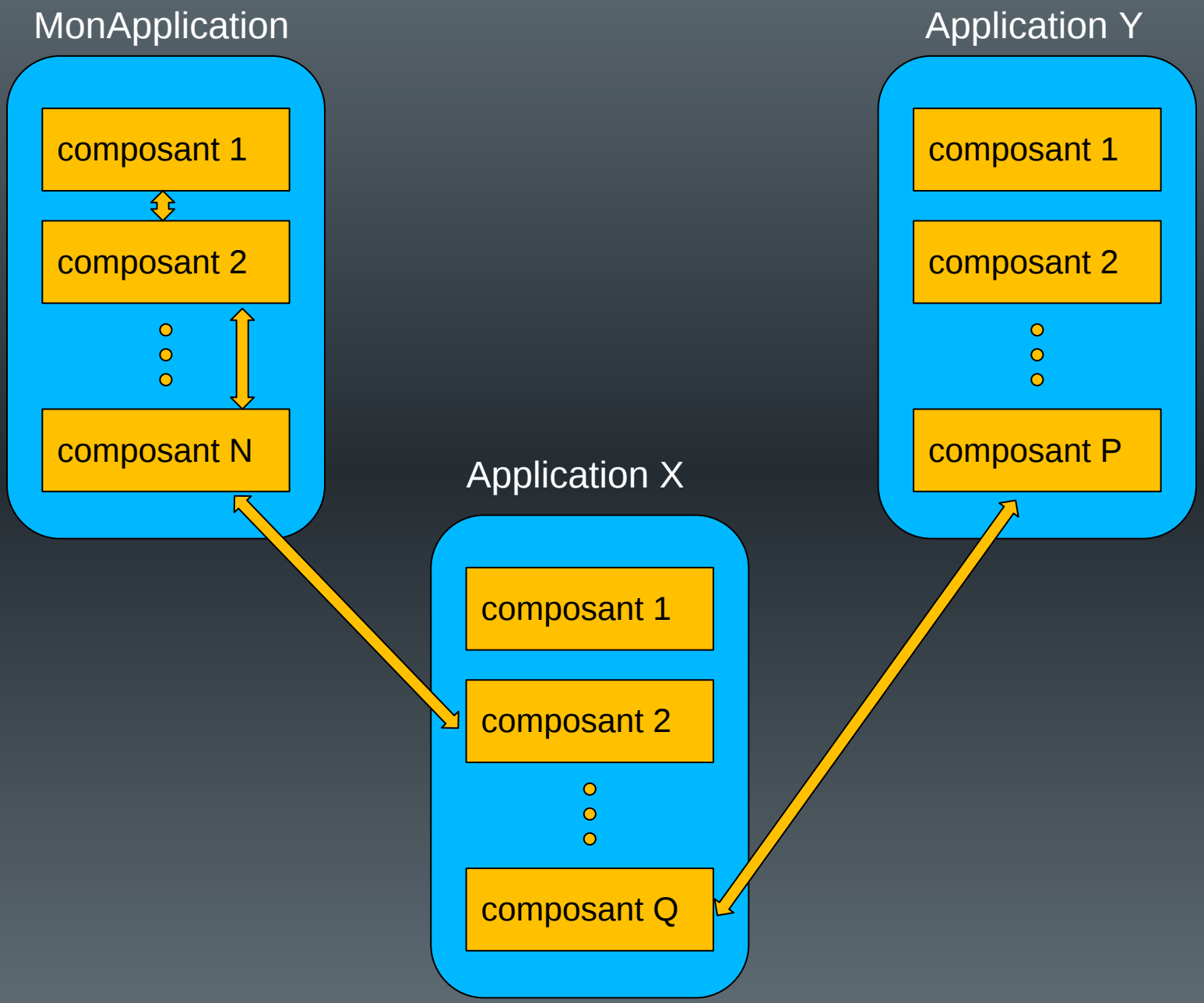
Icônes

Données « constantes »

(sous Android Studio)

Les éléments d'une application

- Une application = {composants}
- Les composants :
  - Existent de manière indépendante
  - Vus comme autant de points d'entrée par le système
    - Pas de « main » dans une application
- Liés au design d'Android :
  - Toute application doit pouvoir démarrer un composant d'une autre application (sous réserve de droits) et récupérer ses « résultats »


MonApplication
composant 1
composant 2
composant N
Application Y
composant 1
composant 2
composant P
Application X
composant 1
composant 2
composant Q

Exemple

- Mon application = application d’effets sur un portrait de l'utilisateur

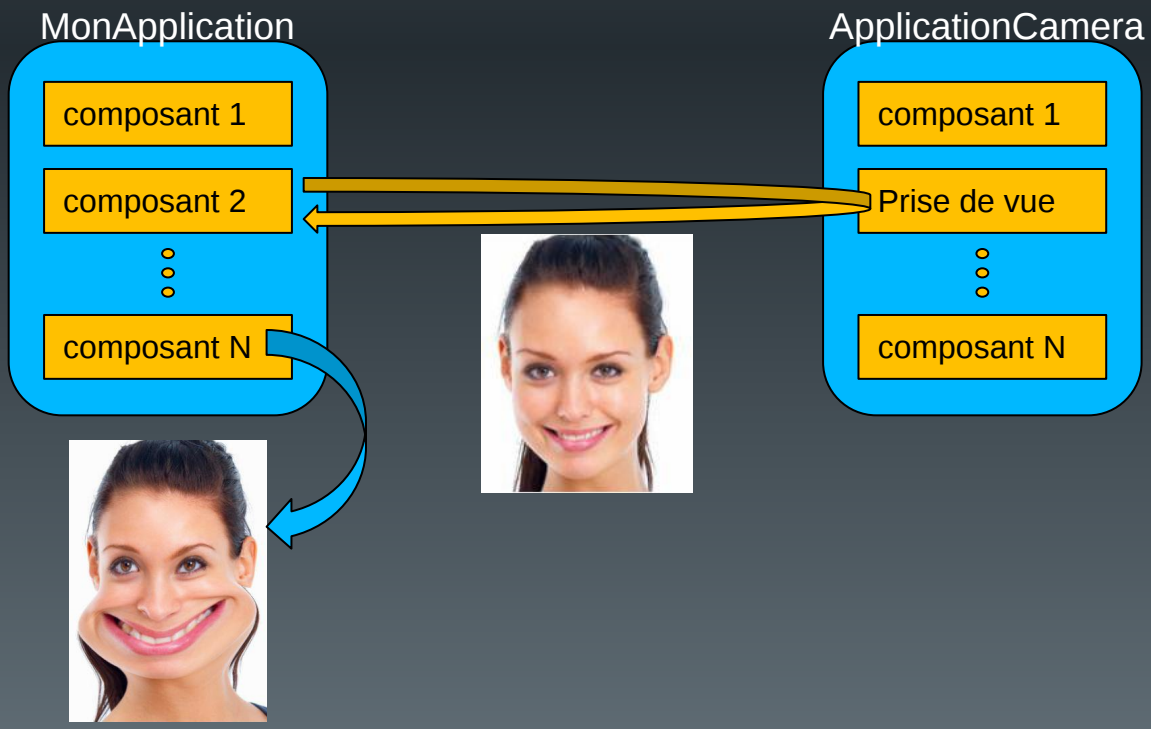
Ma spécialité : traitement d’images

La difficulté : écrire le code de gestion de l'appareil photo embarqué

Source : google play - effets du visage

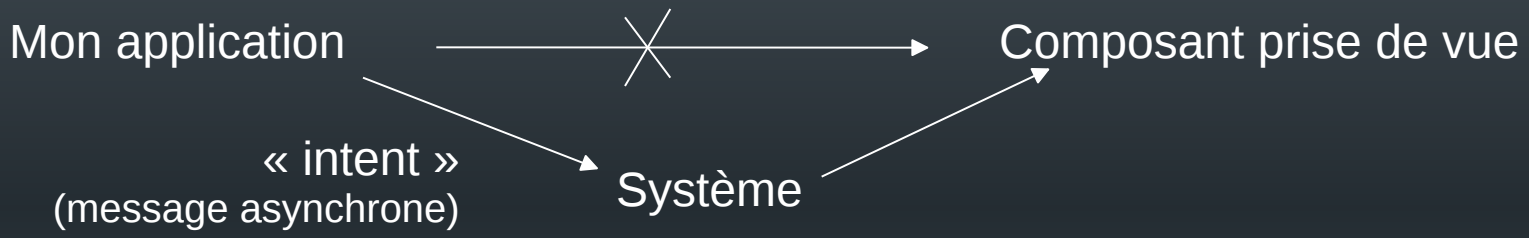
Exemple

- Android :
  - démarrage d'un composant existant permettant la prise de vue
  - Récupération de l'image



Remarques

- Problèmes de droits :



- Problèmes d’information :
  - Le système doit connaître le rôle particulier de certains composants
    - Ce sont les applications qui enregistrent ces informations auprès du système

Les composants

- Les activités (Activity)
  - Un écran avec une interface utilisateur et un contexte
- Les services (Service)
  - Composant sans interface, qui tourne en fond de tâche (lecteur de musique, téléchargement, ...)
- Les fournisseurs de contenu (ContentProvider)
  - I/O sur des données gérées par le système ou par une autre application
- Des récepteurs d'intentions (BroadcastReceiver)
  - Récupération d'informations générales
    - arrivée d'un sms, batterie faible, ...

Les interactions

- Les intentions (*Intent*)
  - Permet d’échanger des informations entre composants
    - Démarrage d’un composant en lui envoyant des données
    - Récupération de résultats depuis un composant
    - Recherche d’un composant en fonction d’un type d’action à réaliser
- Les filtres d'intentions (<intent-filter>)
  - Permet à un composant d'indiquer ce qu'il sait faire
  - Permet au système de sélectionner les composants susceptibles de répondre à une demande de savoir-faire d’une application

AndroidManifest.xml

- Description de l'application
  - Liste des composants
  - Niveau minimum de l'API requise
  - Liste des caractéristiques physiques nécessaires
    - Évite d'installer l'application sur du matériel non compatible (gestion de la visibilité sur Google Play)
  - Liste des permissions dont l'application a besoin
  - Liste des autres API nécessaires
    - ex. Google Map
  - Etc.
- Généré automatiquement par Android Studio

Exemple

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

<application
       android:allowBackup="true"
             android:icon="@mipmap/ic_launcher"
             android:label="@string/app_name"
             android:theme="@style/AppTheme" >
             <activity
             android:name=".MainActivity"
             android:label="@string/app_name" >
             <intent-filter>
                   <action android:name="android.intent.action.MAIN" />
                   <category android:name="android.intent.category.LAUNCHER" />
             </intent-filter>
             </activity>
 </application>
</manifest>
```

Les ressources

- Ressources = toutes les données (autres que le code) utilisées par l'application
- Rangées dans le dossier **res**, puis incluses dans l'apk
  - res/drawable et res/mipmap (images en différentes résolutions)
  - Layout (description en XML des interfaces)
  - Menus (description en XML des menus)
  - Values (définitions en XML des constantes utilisées par l'application : chaînes, tableaux, valeurs numériques, etc.)

strings.xml

- Fichier ressources, contenant toutes les chaînes constantes
  - Principalement utilisées pour l'interface

```
<resources>
    <string name="app_name">MyApplication</string>

    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
</resources>
```

Type de la constante

Nom de la constante (permet l'appel depuis l'application ou un autre fichier XML)

Valeur de la constante

Internationalisation

- Objectif :
  - Disposer de plusieurs versions des textes, libellés, etc utilisés par l'application
  - Choix automatique des textes en fonction de la configuration du périphérique
- Principe
  - Dupliquer le fichier strings.xml : 1 version par langue supportée
  - Stocker chaque version dans un dossier spécifique
    - values-xx (ex. values-en, values-fr, …)
  - Géré via Android Studio

```
app/
  res/
    values/
        strings.xml
    values-en/
        strings.xml
    values-fr/
        strings.xml
```

La classe R

- Classe générée par l'IDE
  - Permet l'accès aux ressources
  - Créée à partir de l'arborescence présente dans le dossier **res**
  - Elle contient des classes internes dont les noms correspondent aux différents types de ressources (drawable, layout, …)
  - Elle contient des propriétés permettant de représenter l'ensemble des ressources de l'application
- Utilisation en Java :
  - R. type.identificateur

```
<resources>
    <string name="app_name">MyApplication</string>

    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
</resources>
```

R.string.app_name

R.string.hello_world

R.string.action_settings

Référencement des ressources en XML

- Forme générale : @type/identificateur

```
<resources>
    <string name="app_name">MyApplication</string>

    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
</resources>
```

@string/app_name

@string/hello_world

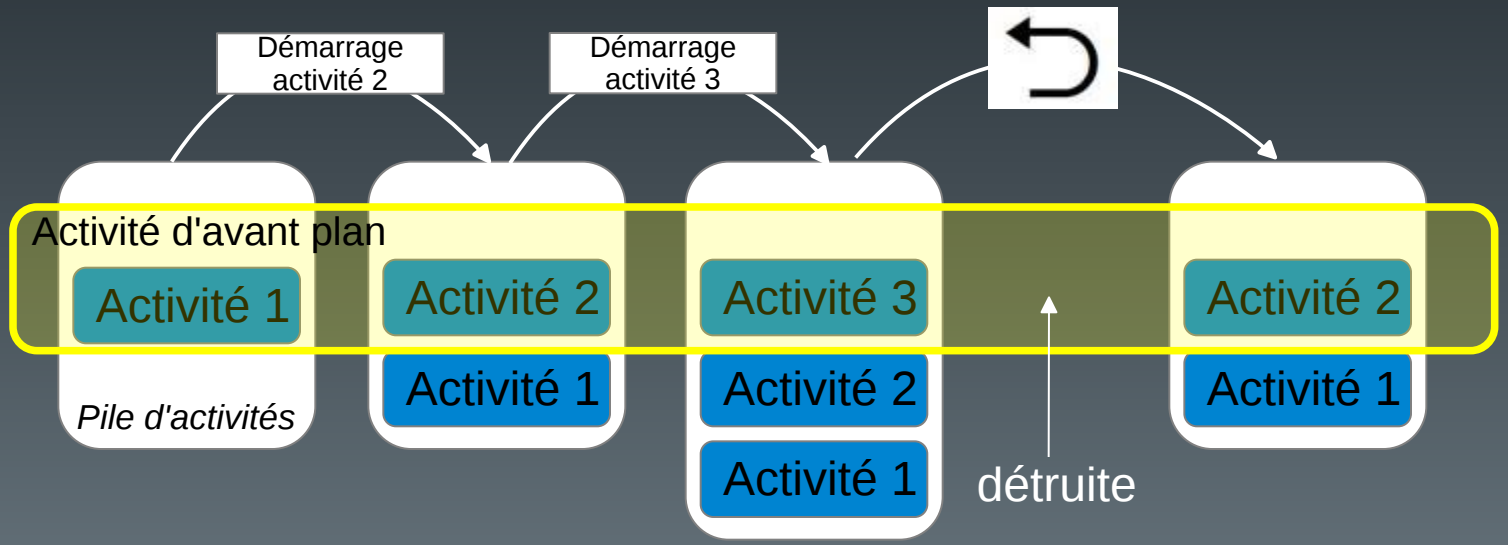
@string/action_settings

Plan du cours

- Introduction
- Architecture d'une application Android
- Les activités
- Définir une interface graphique
- Les intentions explicites
- Les intentions implicites
- Les menus
- Les listes
- Les permissions
- Les content providers

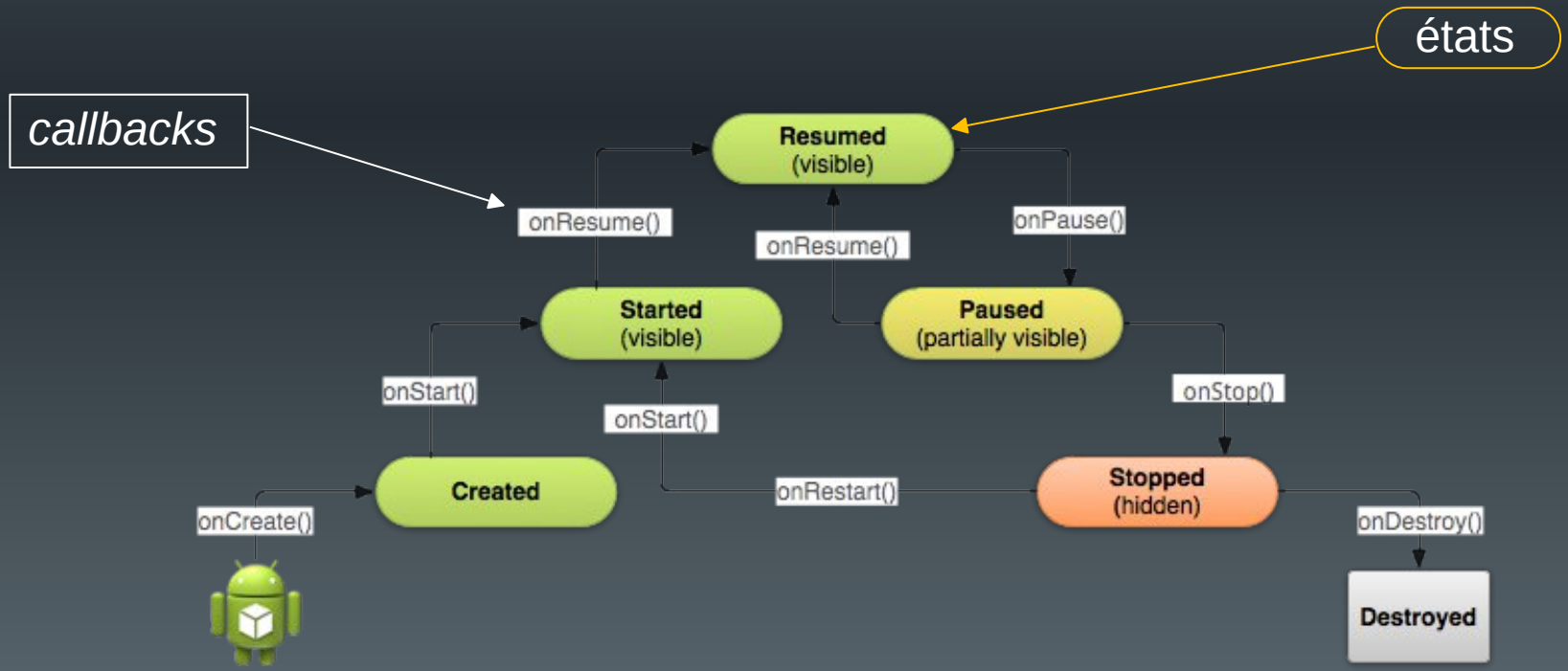
Les activités (1)

- Un composant d'une application, doté d'une interface graphique (IHM) et d'un contexte
- Une activité à la fois visible de l'utilisateur
  - Pour une même application
  - Pour des applications différentes
- Empilement des activités



Les activités (2)

- Cycle de vie
  - Une activité peut se trouver dans différents états en fonction des actions du système et/ou de l'utilisateur :



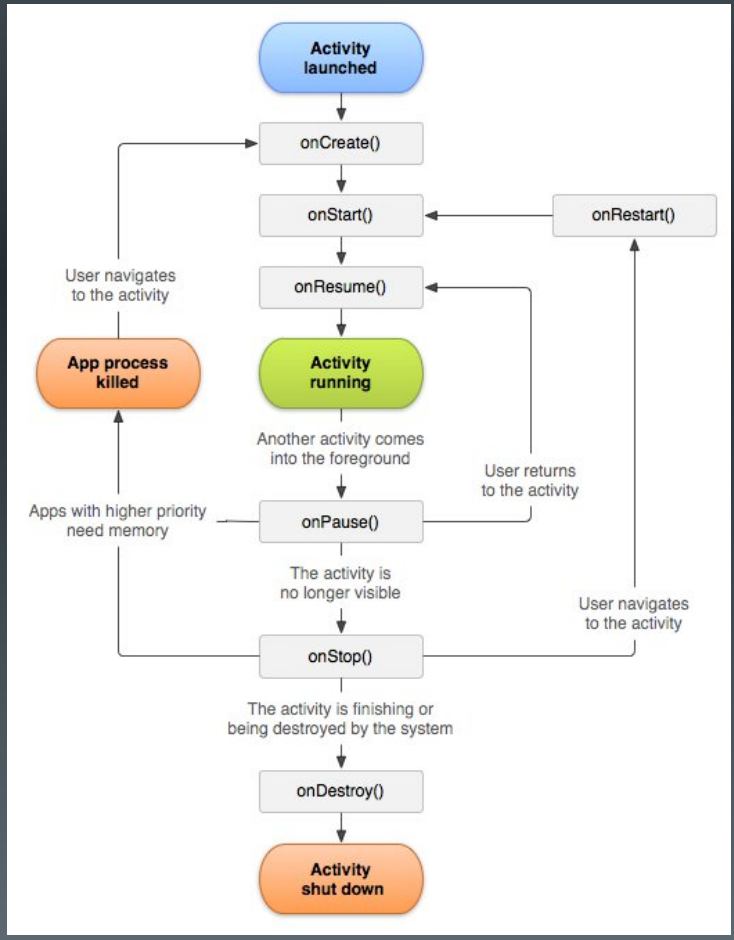
Les activités (3)

- Développement
  - Une classe java par activité ;
  - Les ressources associées (layout, menu, etc.) ;
  - La classe hérite de la classe AppCompatActivity ;
  - Génération d'un code minimum par défaut sous Android Studio.

```
…

public class Bidon extends AppCompatActivity {

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

            // initialisation des variables internes
    …
    }

    …
}
```

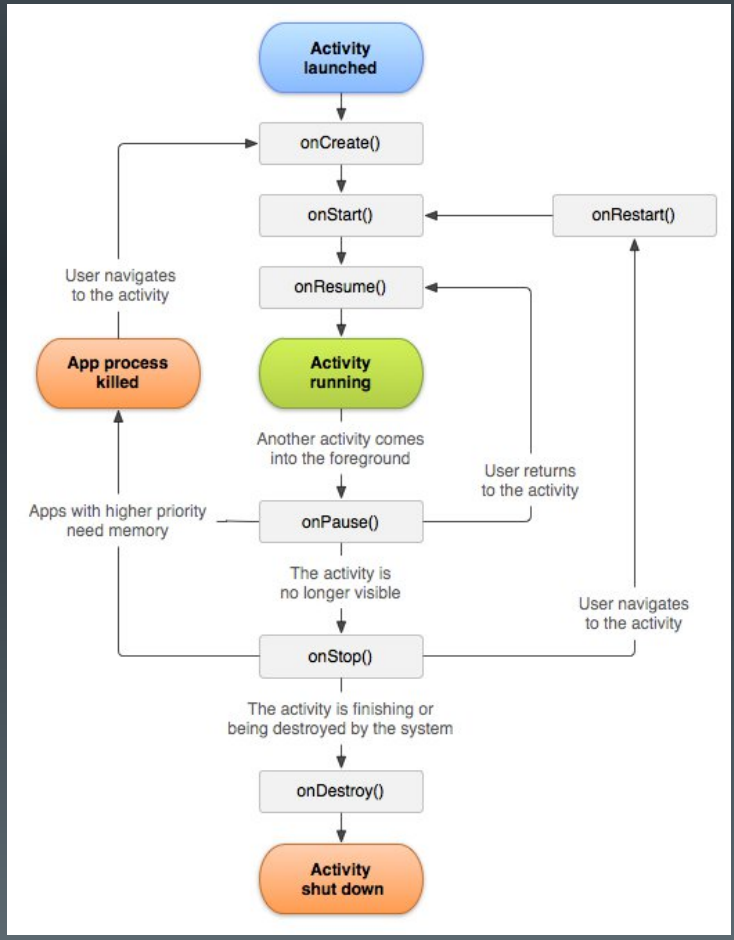
Les activités (4)

- D'autres méthodes peuvent être surchargées, en précisant ce qui doit être fait quand :
  - protected void onDestroy()
    - L'activité se termine
    - Libérer toutes les ressources utilisées
  - protected void onStop()
    - L'activité n'est plus visible
    - Stopper les ressources qui ne sont plus visibles (ex. animations)
  - protected void onPause()
    - L'activité n'est plus au premier plan mais est encore visible (superposition d'une fenêtre de dialogue, multifenêtrage)
    - Stopper les ressources non utilisées (ex. capteur GPS, camera, …)



Les activités (4)

- D'autres méthodes peuvent être surchargées, en précisant ce qui doit être fait quand :
  - protected void onStart()
    - l'activité devient visible
    - Redémarrer les ressources stoppées dans onStop()
  - protected void onResume()
    - l'activité est au premier plan
    - Redémarrer les ressources stoppées dans onPause()
  - protected void onRestart()
    - l'activité redevient visible



Les activités (5)

- Destruction de l'application par le système
  - Cas normal : l'activité est terminée. Le système récupère les ressources, en particulier la mémoire
  - Cas spéciaux : suppression d'une activité non active pour des raisons :
    - de limites des ressources ;
    - de changement d'orientation de l'écran ou de multifenêtrage
  - Le système doit sauvegarder l'état de l'activité, pour pouvoir la redémarrer dans son état courant
    - Sauvegarde dans un objet **Bundle** : couples (nom_donnée, valeur)
      - Contient les données utilisées par l'interface par défaut
      - Systématique dès que l'activité n'est plus visible
    - Surcharge des méthodes de sauvegarde et restauration si d'autres données doivent être sauvées

Les activités (6)

outState

("playerName" , "toto")
("playerPower" , 123.890)
("autreClé01" , autreValeur01)
("autreClé02" , autreValeur02)
…

- Sauvegardes
  - **void onSaveInstanceState(Bundle outState)**

```
public final static String PLAYER_NAME_KEY = "playerName"
public final static String PLAYER_POWER_KEY = "playerPower"
```

Constantes définissant le nom des clés

```
String nomJoueur;
float  puissance;
```

Variables internes

```
// Fonction callback appelée en cas de destruction temporaire de l'activité
@Override
public void onSaveInstanceState(Bundle outState) {

    outState.putString(PLAYER_NAME_KEY, nomJoueur);
    outState.putFloat(PLAYER_POWER_KEY, puissance);

    // appel à la super classe pour sauvegarder les données de l'interface
    super.onSaveInstanceState(outState);
}
```

Sauvegarde de la valeur des variables internes dans le bundle

Les activités (7)

- Récupération
  - **void onCreate(Bundle savedInstanceState)**

```
@Override
public void onCreate(Bundle savedInstanceState) {
    // appel à la super classe pour mettre à jour les données de l'interface
    super.onCreate(savedInstanceState);

    // récupération des données internes
    if (savedInstanceState != null) {
        nomJoueur = savedInstanceState.getString(PLAYER_NAME_KEY);
        puissance = savedInstanceState.getFloat(PLAYER_POWER_KEY);
}
```

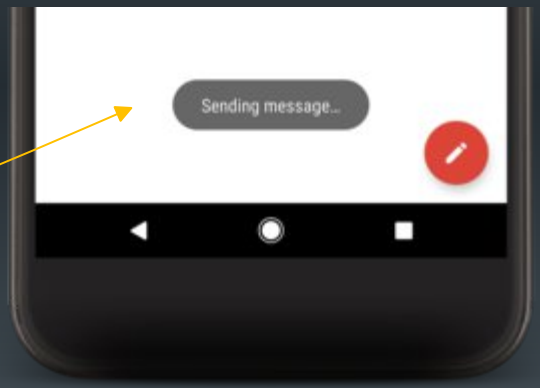
  - **void onRestoreInstanceState (Bundle savedInstanceState)**

```
// Callback appelé après onStart() uniquement si un appel à onSaveInstanceState()
//  a été fait. Le bundle savedInstanceState est le même que pour onCreate()
@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
        nomJoueur = savedInstanceState.getString(PLAYER_NAME_KEY);
        puissance = savedInstanceState.getFloat(PLAYER_POWER_KEY);
}
```

Remarques

https://www-lisic.univ-littoral.fr/~renaud/

- Affichage de messages de mises au point
  - Possibilité d'utiliser *System.out.println*
    - Affichage dans la console d'Android Studio
  - La classe Toast



Source : develop.android.com

Le message

Toast.*makeText*(**this**, **"Sending message…"**, Toast.***LENGTH_SHORT***).show();

Le contexte

La durée d'affichage

Le déclenchement de l'affichage

Plan du cours

- Introduction
- Architecture d'une application Android
- Les activités
- Définir une interface graphique
- Les intentions explicites
- Les intentions implicites
- Les menus
- Les listes
- Les permissions
- Les content providers

Quelques règles de base

- Interface = seul contact de l'utilisateur
  - Faire attirant
  - Faire simple
    - L'application doit être intuitive
    - Éviter les trop longs messages
- Faire ergonomique
  - L'enchaînement des activités doit être rapide
  - L'utilisateur doit toujours connaître l'état courant de l'activité
- Conseils et « matériels » :
  - http://developer.android.com/design

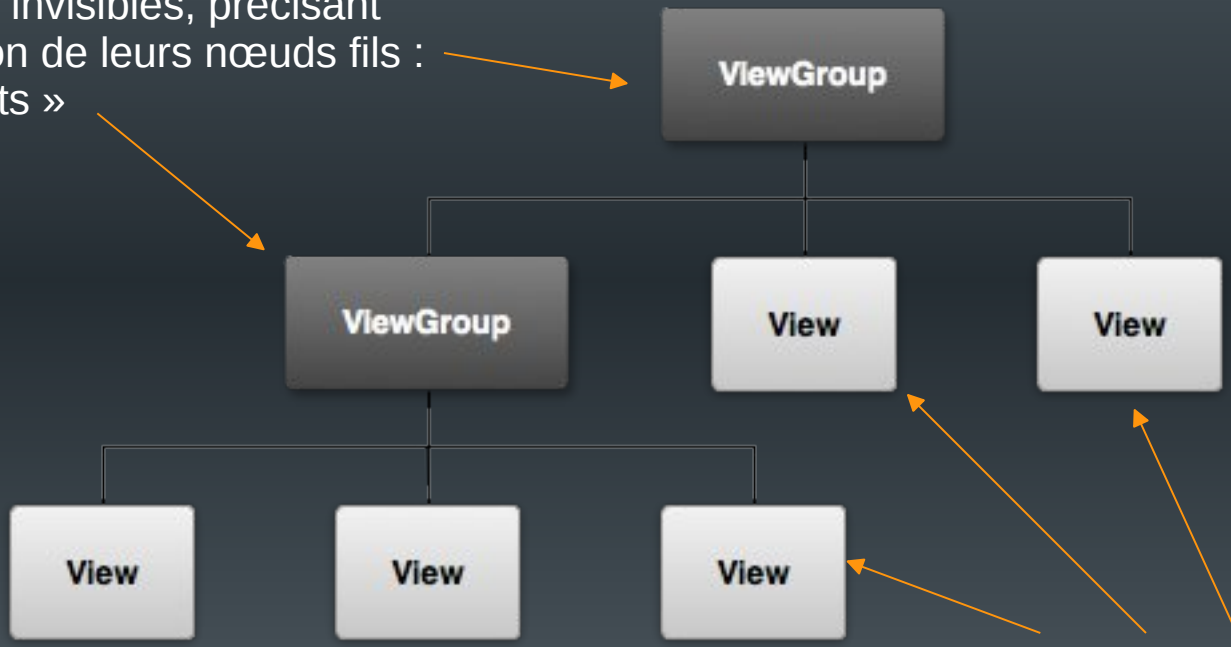
Définir une interface graphique

- Définir les « interacteurs »
  - Objets graphiques visibles par l'utilisateur pour :
    - L'affichage (texte, images, etc.)
    - L'interaction (boutons, cases, champs de saisie, etc.)
- Définir leur mise en page
  - Positions dans l'interface (fixes ou relatives)
- XML ou Java (sauf traitement de l'interaction : Java seul)
    - Privilégier XML
      - Souplesse de mise à jour
      - Permet la prise en compte simplifiée de différents types d'écran

Représentation d'une interface

- Représentation arborescente

Conteneurs invisibles, précisant l'organisation de leurs nœuds fils : les « Layouts »



Source : developer.android.com

exemple

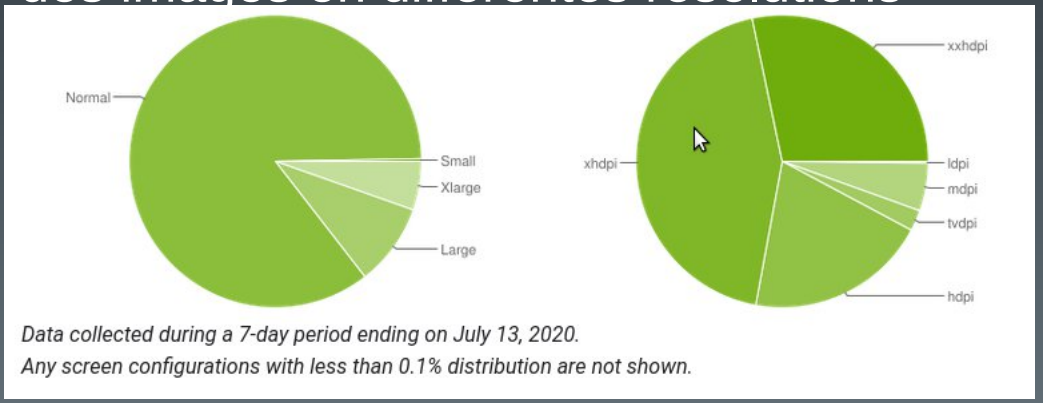
Objets graphiques permettant l'interaction (boutons, zones de texte, etc.) : les Widgets

Les Layouts (1)

- Zone invisible assurant l'organisation automatique des composants graphiques
  - Peuvent être déclarées en XML ou Java
  - Privilégier XML
    - Séparation du code et de la mise en page
    - Souplesse d'adaptation à différents périphériques
- Possèdent des propriétés « intuitives » permettant l'organisation des composants
- Nombreux layouts différents
  - Peuvent être imbriqués (cf arborescence)
- Un layout doit être chargé dans onCreate()
  - setContentView(R.layout.nom_du_layout)

Les Layouts (2)

- Gestion multi-écrans
  - Différentes tailles
    - small, normal, large, xlarge
  - Différentes densités de pixels
    - ldpi (120dpi), mdpi (160 dpi), hdpi (240 dpi), xhdpi (320 dpi), xxhdpi (480 dpi), xxxhdpi (640 dpi)
  - Prévoir un layout par taille (et orientation) de l'écran si nécessaire
    - effets de positionnements relatifs pouvant être gênants
  - Prévoir des images en différentes résolutions



Source
developer.android.com

Les Layouts (3)

- Fonctionnement similaire à l'internationalisation
  - Un sous-dossier spécifique à chaque layout et/ou à chaque image

```
MyProject/
    res/
        layout/              # default (portrait)
            main.xml
        layout-land/         # landscape
            main.xml
        layout-large/        # large (portrait)
            main.xml
        layout-large-land/   # large landscape
            main.xml
```

```
MyProject/
    res/
        drawable-xhdpi/
            awesomeimage.png
        drawable-hdpi/
            awesomeimage.png
        drawable-mdpi/
            awesomeimage.png
        drawable-ldpi/
            awesomeimage.png
```

Source : developer.android.com

LinearLayout (1)

- Aligne les nœuds dans une seule direction
  - horizontale (par défaut)
  - verticale

activity_main.xml | MainActivity.java

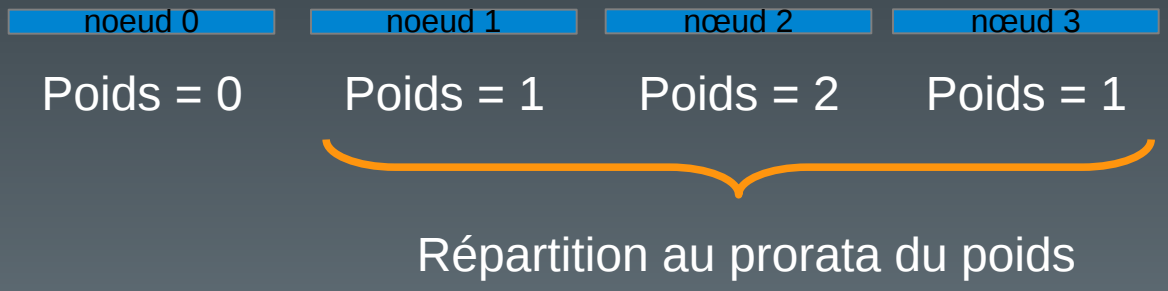
```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      android:orientation="vertical"
8      tools:context=".MainActivity">
9
10     <TextView
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:text="Hello World!" />
14
15     <TextView
16         android:id="@+id/textView"
17         android:layout_width="match_parent"
18         android:layout_height="wrap_content"
19         android:text="Coucou" />
20
21 </LinearLayout>
```

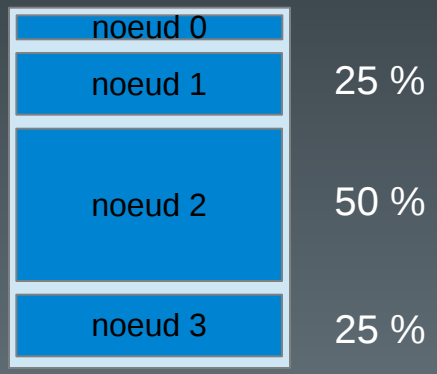
Taille du composant :

- match_parent : S'adapte à la taille du conteneur parent (ici l'écran)
- wrap_content : S'adapte à la taille de ce qu'il contient (ici une zone de texte)
- *dimension fixe*

LinearLayout (2)

- Modification du « poids » de chaque nœud
  - Permet de changer la taille de la zone occupée par chaque nœud dans l'écran
  - Ajout d'un attribut android:layout_weight à chaque nœud
    - 0 (par défaut) : n'utilise que la zone nécessaire au nœud
    - n>0 : poids du nœud par rapport aux autres nœuds





LinearLayout (3)

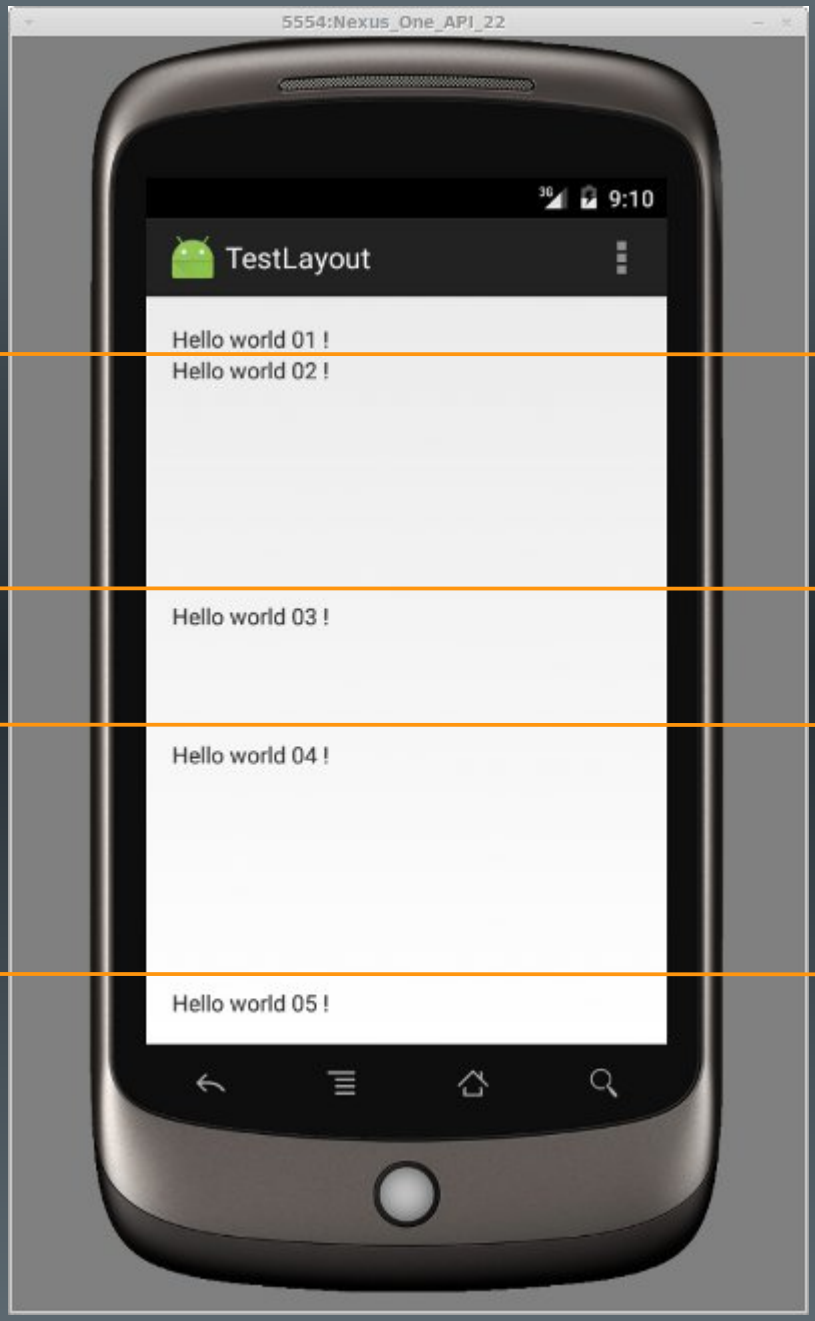
- Exemple

```
<TextView
    android:text="@string/hello_world_01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />

<TextView
    android:text="@string/hello_world_02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="2"
    />
<TextView
    android:text="@string/hello_world_03"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    />
<TextView
    android:text="@string/hello_world_04"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="2"
    />
<TextView
    android:text="@string/hello_world_05"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />
```

40 %

20 %

40 %



LinearLayout (4)

- Alignement de chaque noeud dans sa zone
  - Ajout d'un attribut android:layout_gravity
  - Nombreuses valeurs possibles :
    - center, center_vertical, center_horizontal
    - left, right, top, bottom
    - Etc. (cf LinearLayout.LayoutParams)

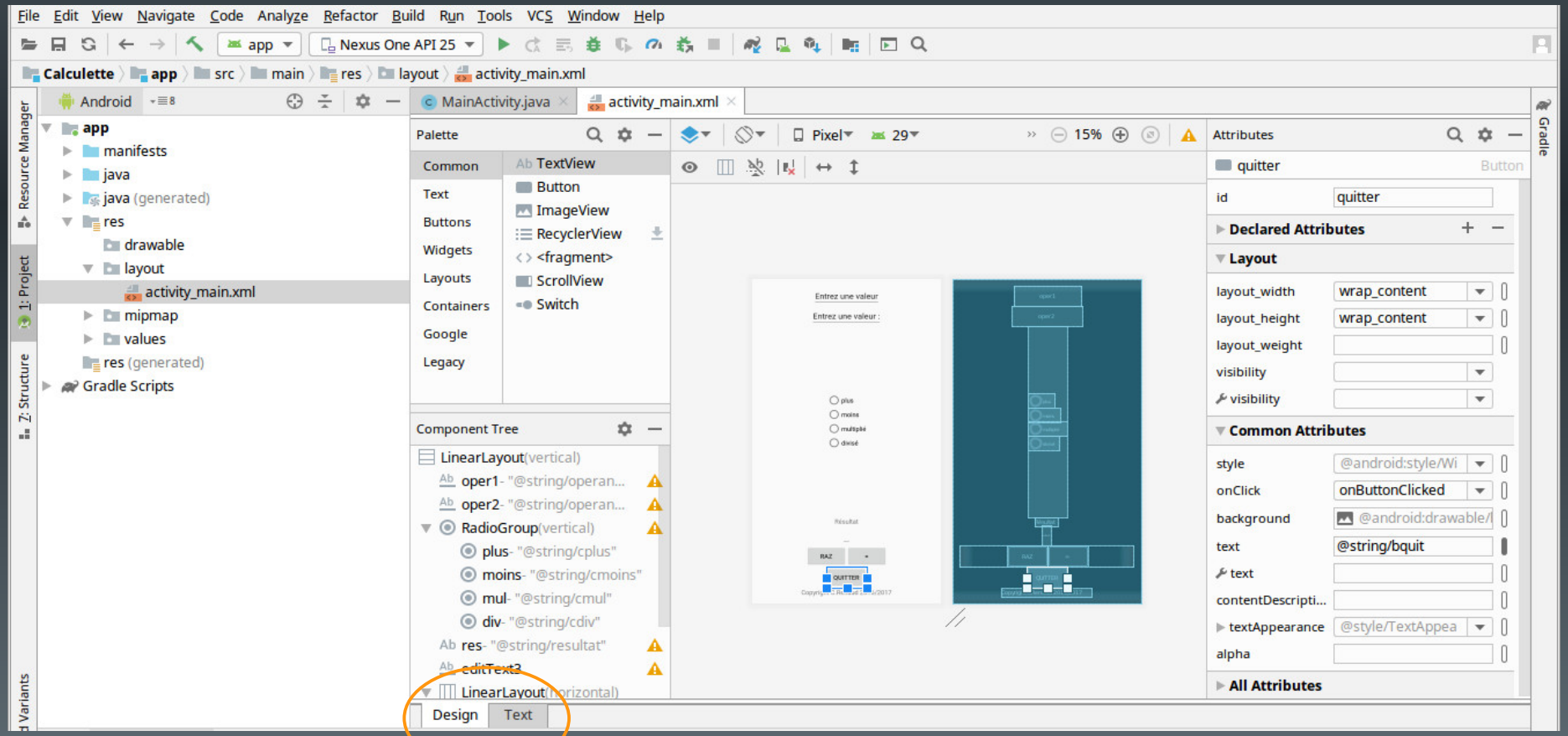
LinearLayout (5)

- Exemple

```
<TextView
    android:text="@string/hello_world_01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />

<TextView
    android:text="@string/hello_world_02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_weight="2"
    />
<TextView
    android:text="@string/hello_world_03"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="right"
    android:layout_weight="1"
    />
<TextView
    android:text="@string/hello_world_04"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:gravity="center"
    android:layout_weight="2"
    />
<TextView
    android:text="@string/hello_world_05"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="right"
    />
```

Remarque

- Possibilité d'organiser visuellement les layouts sous Android Studio

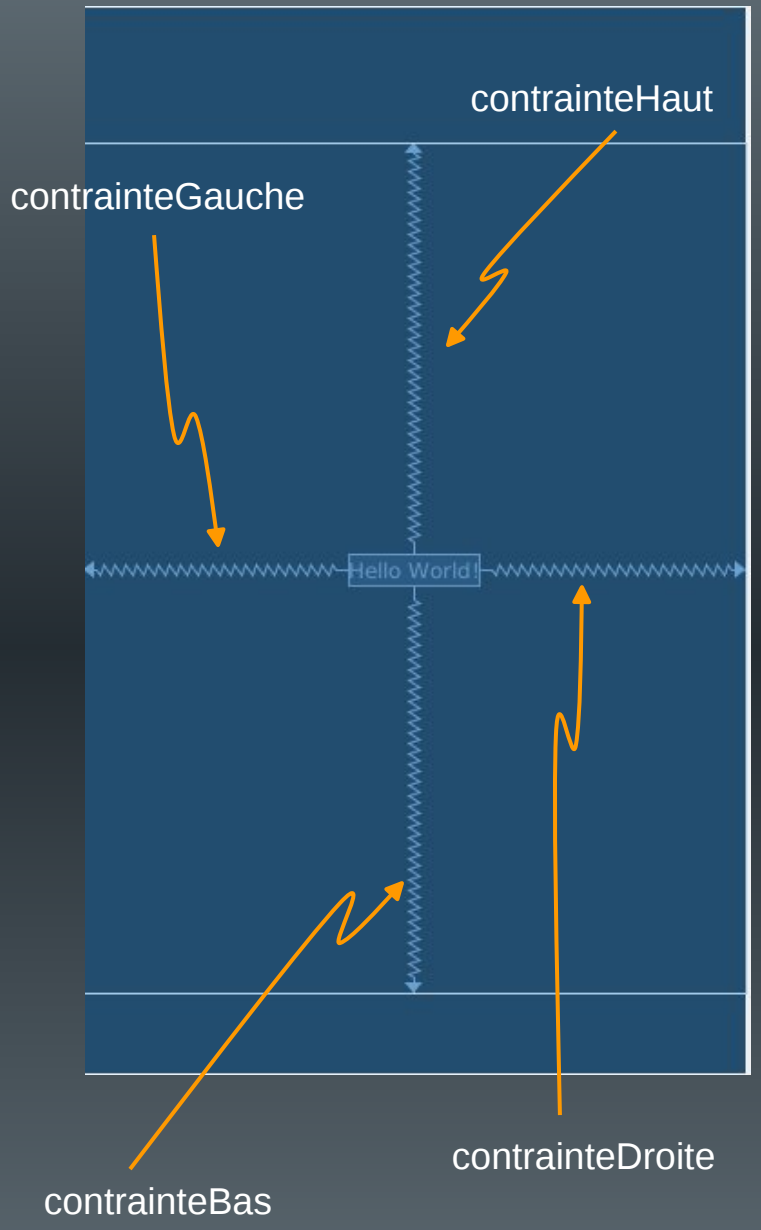


ConstraintLayout (1)

- Objectif :
  - Faciliter la création de layouts complexes
  - Réduire l'imbrication de layouts
    - Consommatrice de ressources
- Organisation intuitive :
  - Création de relations entre les composants graphiques et avec leur layout parent
    - Notion de contraintes
  - Très flexible
  - Bien adapté à l'éditeur graphique

ConstraintLayout (2)

- Contraintes
  - Au moins une contrainte horizontale
  - Au moins une contrainte verticale

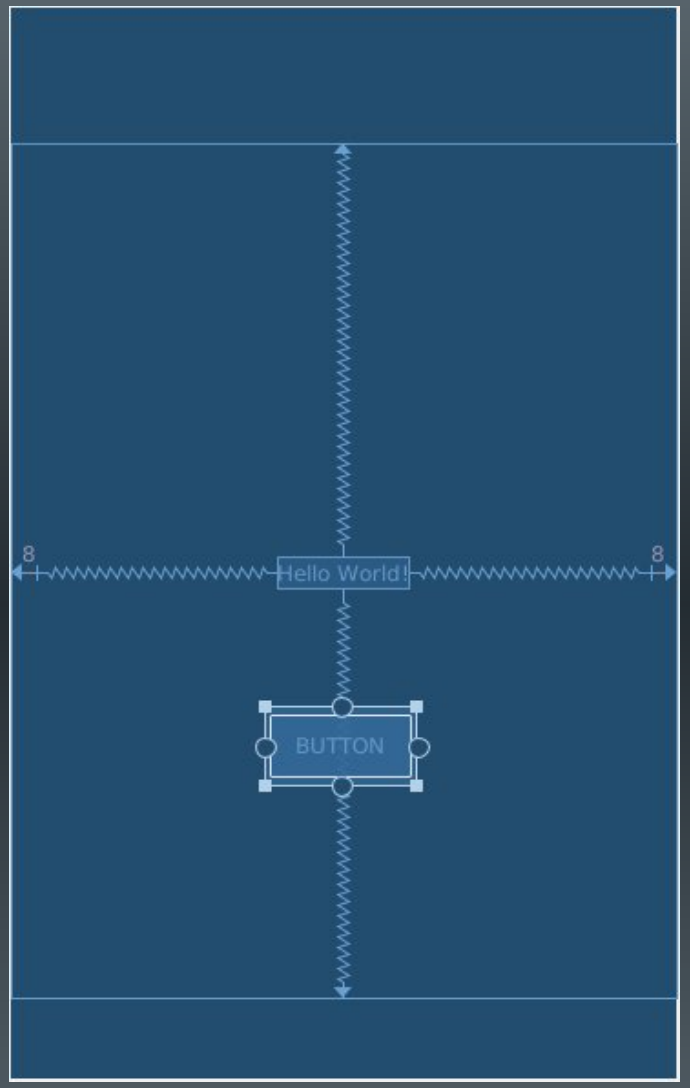


Exemple :

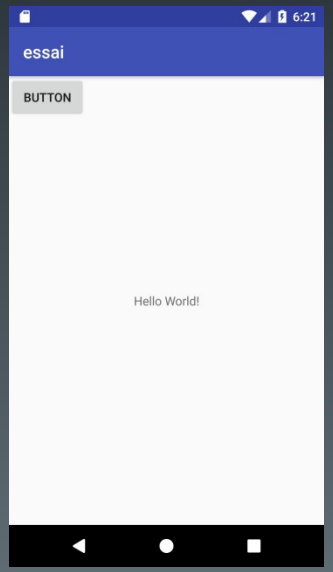
```
<android.support.constraint.ConstraintLayout
xmlns:android=
    "http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

ConstraintLayout (2)

- Remarque
  - Placement sans contrainte d'un composant graphique via l'éditeur graphique
    - Apparaît au bon endroit dans l'éditeur
  - Placé par défaut en (0,0) lors de l'exécution …



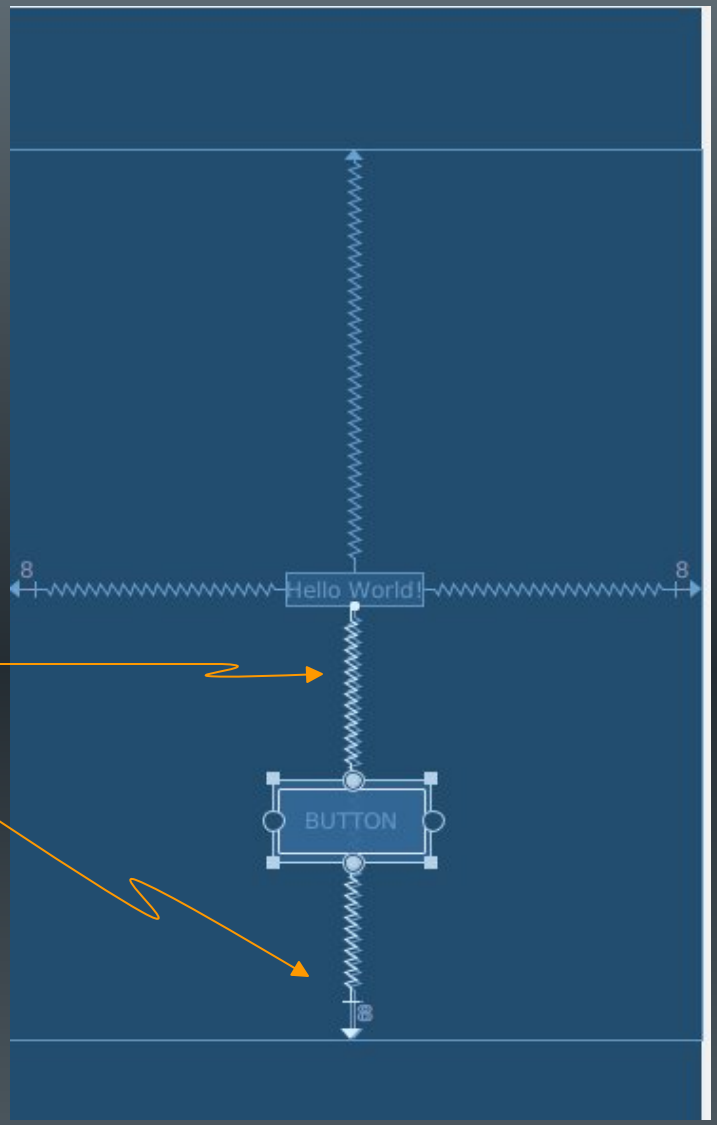

```
<Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        tools:layout_editor_absoluteX="148dp"
        tools:layout_editor_absoluteY="373dp" />
```



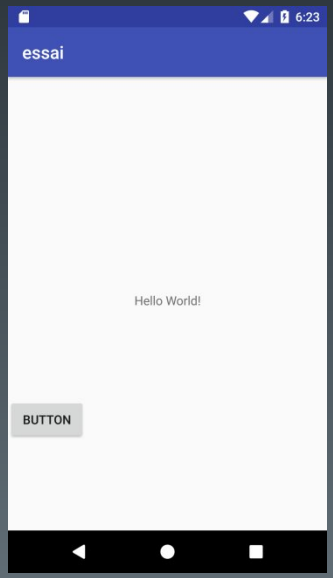
ConstraintLayout (3)

- Ajout d’une contrainte verticale

```
<Button
      android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginBottom="8dp"
      android:layout_marginTop="8dp"
      android:text="Button"
      app:layout_constraintTop_toBottomOf="@+id/textView "
      app:layout_constraintBottom_toBottomOf="parent"
     tools:layout_editor_absoluteX="148dp" />
```



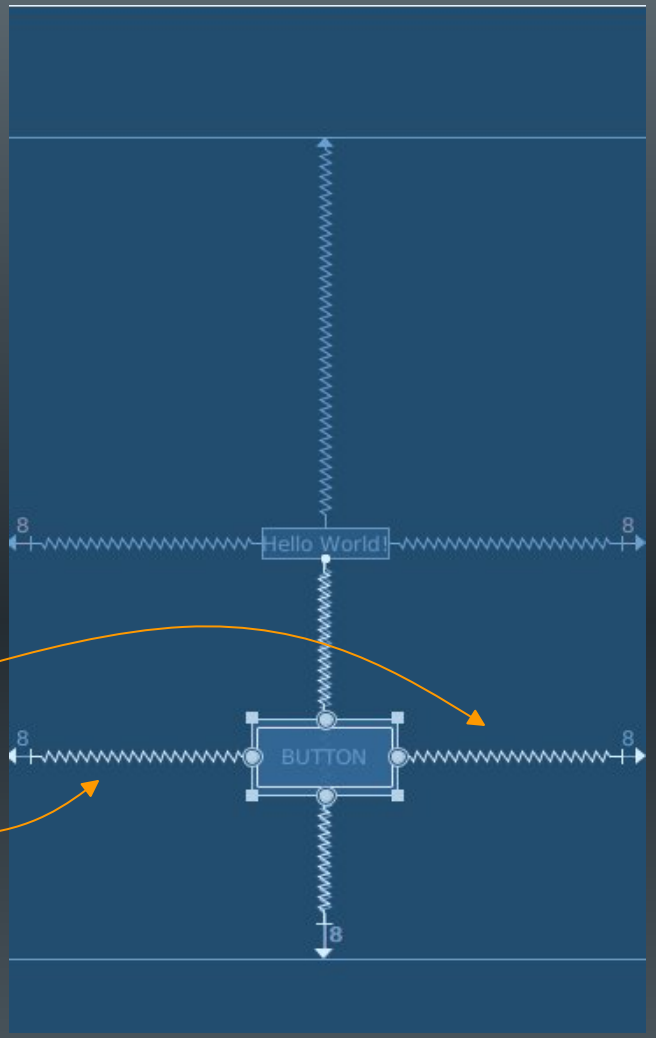
X=0

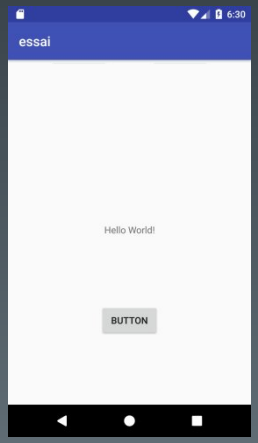


ConstraintLayout (4)

- Ajout d'une contrainte horizontale

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:text="Button"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView" />
```



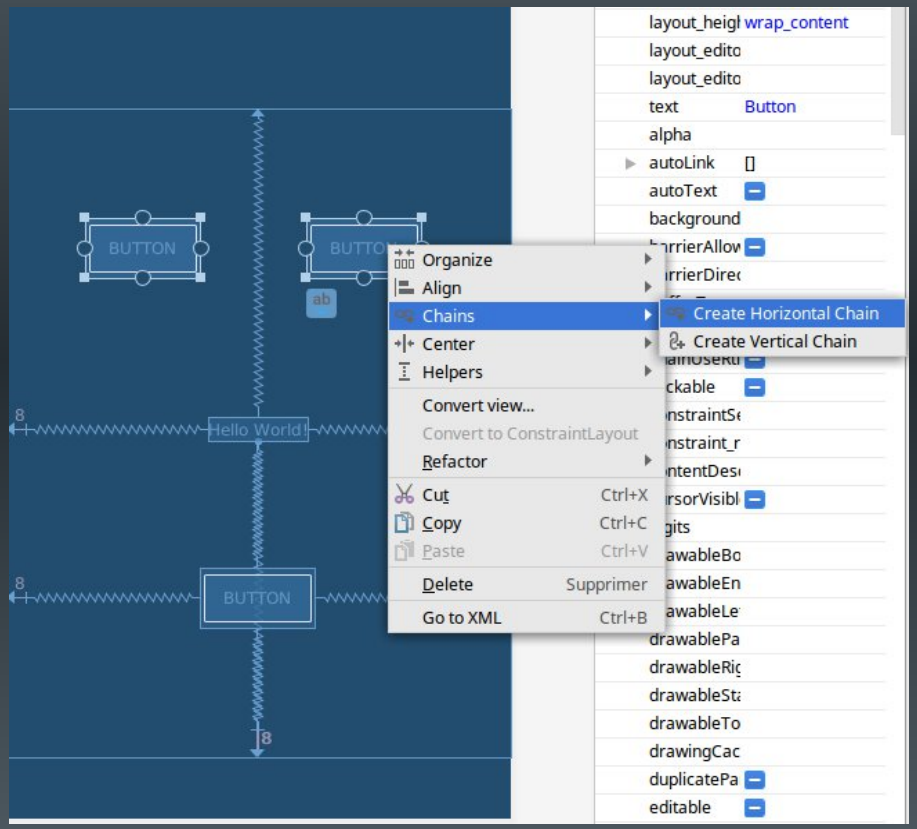


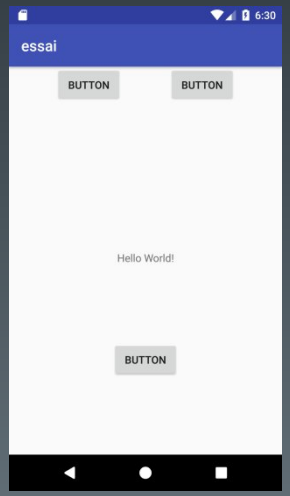
Remarque :

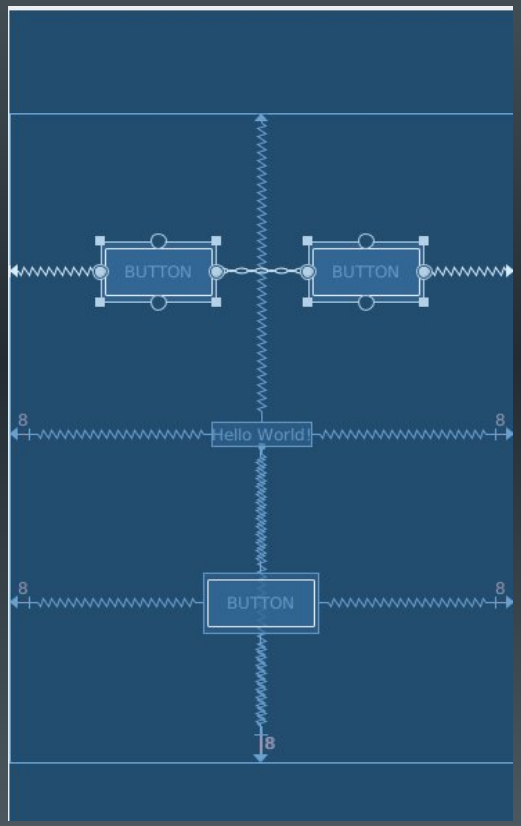
- `@+id/nom` rajoute nom à la classe R uniquement si `nom` n'existe pas
- Utile si le composant nommé est défini après dans le fichier xml

ConstraintLayout (5)

- Création de chaînages

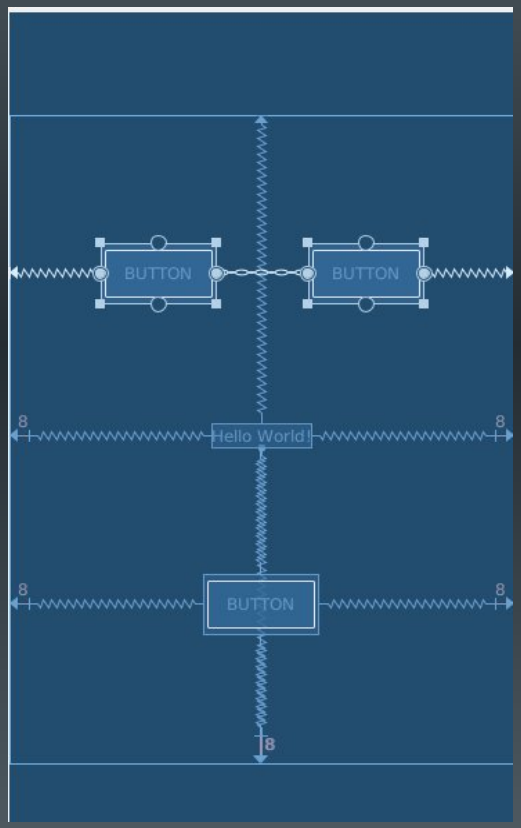






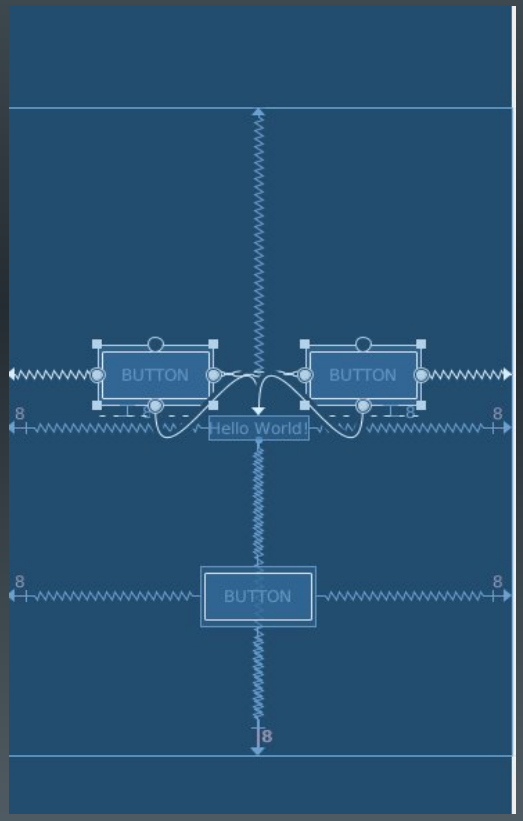
Pas de contraintes verticales
(y = 0)

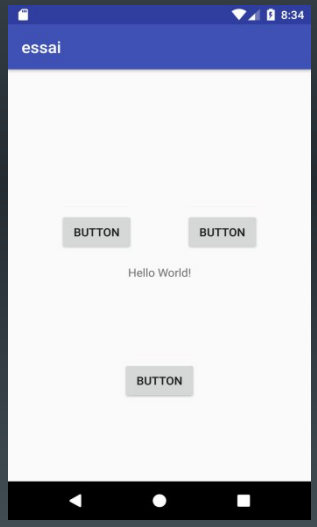
ConstraintLayout (6)

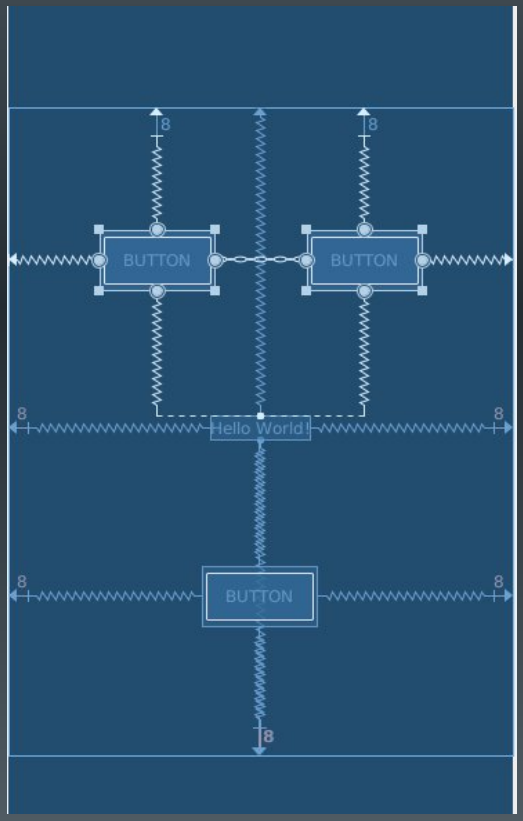


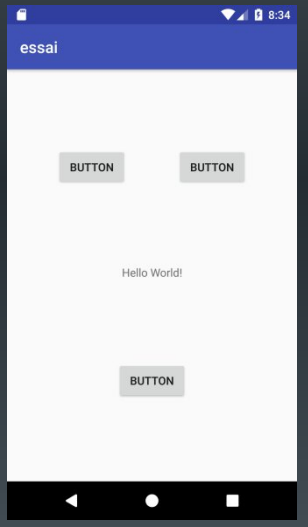

```
<Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        app:layout_constraintEnd_toStartOf="@+id/button4"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        tools:layout_editor_absoluteY="101dp" />

    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toEndOf="@+id/button3"
        tools:layout_editor_absoluteY="101dp" />
```

ConstraintLayout (7)









Les Widgets

- Composants graphiques visibles par l'utilisateur
  - Widgets simples : zones de texte, boutons, listes, etc.
  - Widgets plus complexes : horloges, barres de progression, etc.
- Héritent de la classe View
- Utilisation :
  - Définition en XML (type, taille, centrage, position, etc.)
  - Comportement en Java
  - Peuvent également être créés dynamiquement en Java

Les TextView

- Widget permettant l'affichage d'un texte
  - Normalement non éditable
- Exemple :

```
<TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/letexte"
        android:hint="texte initial"
        android:layout_gravity="center"
        android:gravity="center"
/>
```

- Nombreux autres attributs
  - Cf classe TextView

Les EditText

- Widget permettant la saisie d'un texte (TextFields)
  - Accès : ouverture d'un clavier pour la saisie
  - nombreux attributs permettant l'aide à la saisie

```
<EditText
    android:id="@+id/email_address"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/email_hint"
    android:inputType="textEmailAddress" />
```

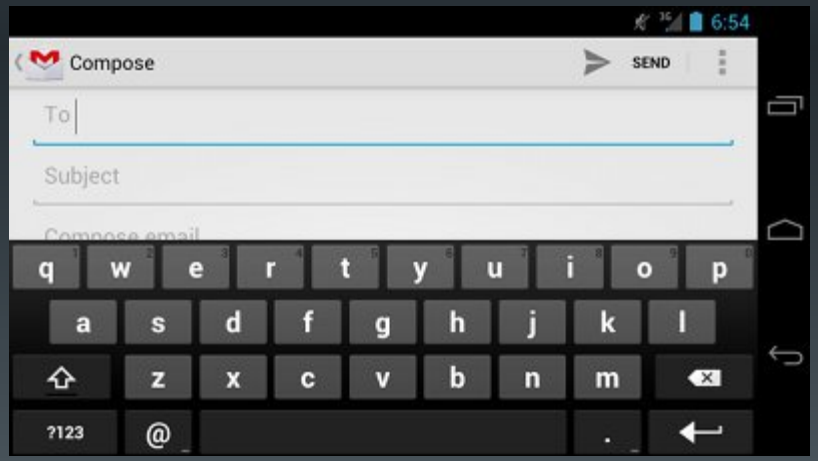
"text" : Normal text keyboard.

"textEmailAddress" : Normal text keyboard with the @ character.

"textUri" : Normal text keyboard with the / character.

"number" : Basic number keypad.

"phone" : Phone-style keypad.



Source : developer.android.com

Les Button

- Widget représentant un bouton d'action
  - Renvoie un événement lors de l'appui
  - Peut contenir un texte, une image ou les deux
- Exemples :

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    ... />
```

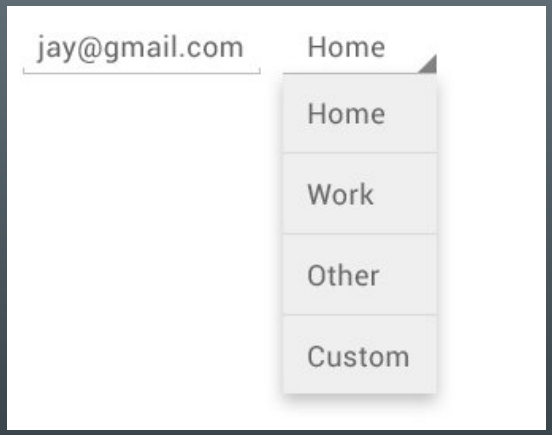
```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:drawableLeft="@drawable/button_icon"
    ... />
```


```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon"
    ... />
```

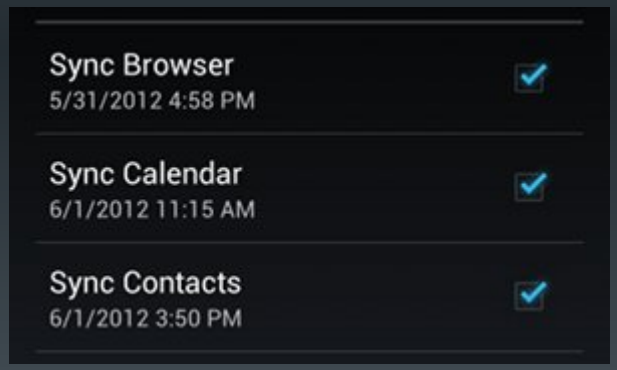
Source : developer.android.com

En vrac ...

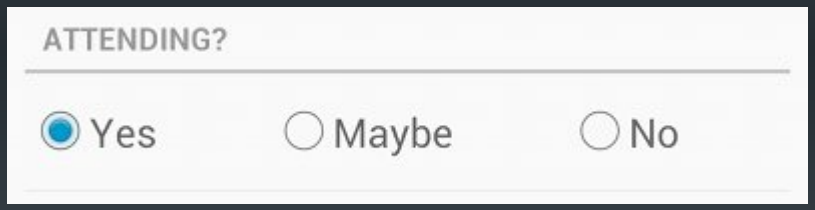
- Quelques autres widgets
  - Source developer.android.org



Spinner



CheckBox



RadioButton

ToggleButton

Switch
(android 4.0+)

Implantation du comportement (1)

- Les fichiers XML ne permettent que de :
  - positionner les composants ;
  - définir leurs caractéristiques.
- Nécessité de :
  - définir leur comportement
    - type d'interaction (clic court, clic long, etc.)
    - code de prise en compte (Java)
  - lier composant et code (XML ou Java)
    - XML : attribut android:onClick
    - Java : instancier un *event listener*

Implantation du comportement (2)

- Attribut android:onClick
  - Doit être suivi du nom de la méthode à appeler en cas de déclenchement
  - Prototype :
    - public void nomDeLaMethode(View maVue)

```
<Button
        android:id="@+id/monBouton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
        android:text="@string/monTexte"
    android:onClick="onBoutonClique"
/>
```

```
public void onBoutonClique(View maVue) {

    System.out.println("le bouton a été cliqué") ;

}
```

Permet de récupérer des informations sur le composant graphique qui a généré l'événement
Initialisé par le système avant l'appel

Récupération :

maVue.getId() → R.id.monBouton

Implantation du comportement (3)

- Les *event listener*
  - interfaces de la classe View
  - ne disposent que d'une seule méthode à implanter
  - méthode appelée quand le composant associé est déclenché par l'utilisateur
- Exemples :

Interface	Méthode
View.OnClickListener	abstract void onClick(View v)
View.OnLongClickListener	abstract boolean onLongClick(View v)
View.OnFocusChangeListener	abstract void onFocusChange(View v, boolean hasFocus)

Implantation du comportement (3)

- Exemple : l'interface View.onClickListener
  - public void onClick(View v)

Retrouver un widget à partir du nom qui lui a été associé dans le xml

```
...
Button button = (Button) findViewById(R.id.button_name);

button.setOnClickListener(new View.OnClickListener() {

    public void onClick(View v) {
        // Do something in response to button click
    }

});
...
```

Création d'un « OnClickListener »

Source : developer.android.com

Surcharge de la méthode « onClick » de l'interface « onClickListener »

Associer un « onClickListener » au bouton

TP 02

Plan du cours

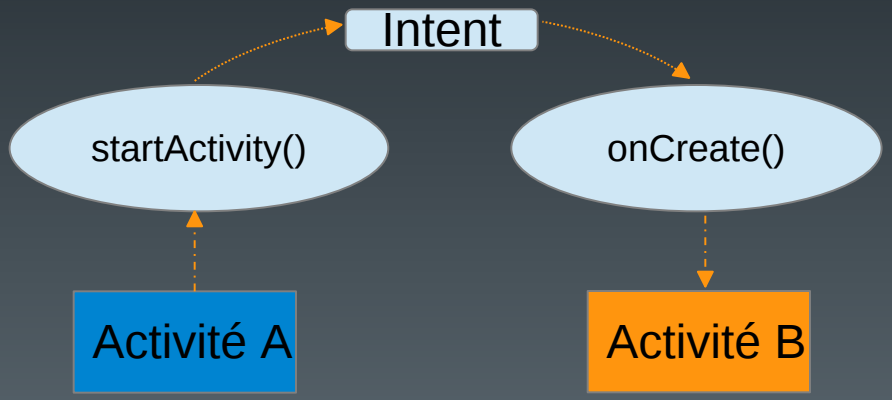
- Introduction
- Architecture d'une application Android
- Les activités
- Définir une interface graphique
- Les intentions explicites
- Les intentions implicites
- Les menus
- Les listes
- Les permissions
- Les content providers

Les Intentions

- Classe représentant un message échangé entre une activité et un composant présent sur le système
  - Une autre activité
  - Un service
  - Un diffuseur d'événements
- Deux types de messages
  - Explicite : on nomme le composant à démarrer
  - Implicite : on demande au système de trouver un composant adéquat, en fonction d'une action à effectuer

Les Intentions explicites

- Message adressé à un composant connu
  - On donne le nom de la classe correspondante
  - Généralement réservé aux composants appartenant à la même application …
- Le composant est démarré immédiatement



Création et lancement

Création d'une intention

```
Intent intention = new Intent (…) ;
…
startActivity(intention) ;
```

Démarrage d'une activité

```
Intent intention = new Intent (…) ;
…
startService(intention) ;
```

Démarrage d'un service

```
void maFonction(...) {
  ...
  Intent intention = new Intent (…) ;
  …
  startActivityForResult(intention, requestCode) ;
  …
}
```

Déprécié depuis android 12

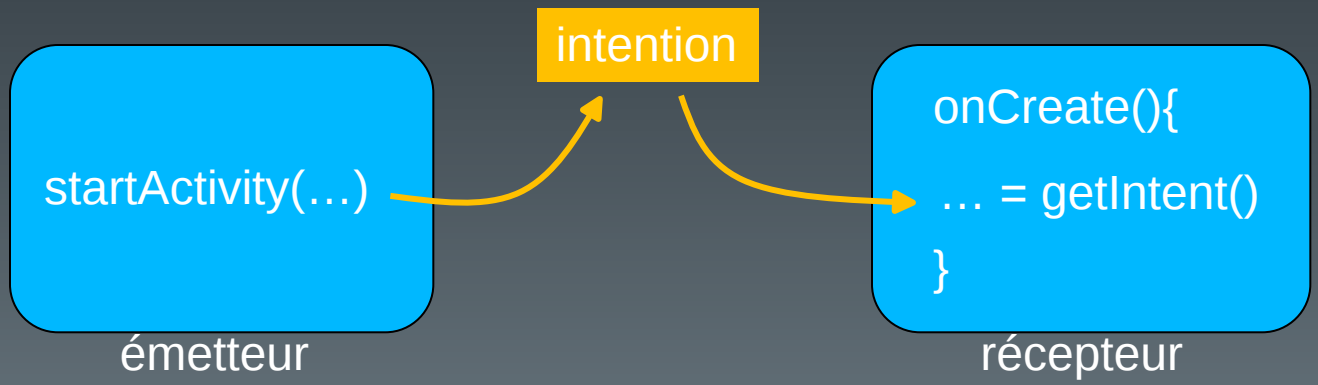
Démarrage d'une activité avec attente d'un résultat, récupéré dans la surcharge de cette méthode

```
protected void onActivityResult (int requestCode, int resultCode, Intent data){
  …
}
```

Récupération

Code côté **récepteur**

```
public void onCreate (Bundle savedInstanceState) {
    ...
    // récupération de l'intention
    Intent intention = getIntent () ;
    ...
    // extraction des informations reçues
    ...
    // traitement des informations reçues
    ...
}
```



Intentions avec résultats (émetteur)

```
public static final int CODE = 4 ;
```

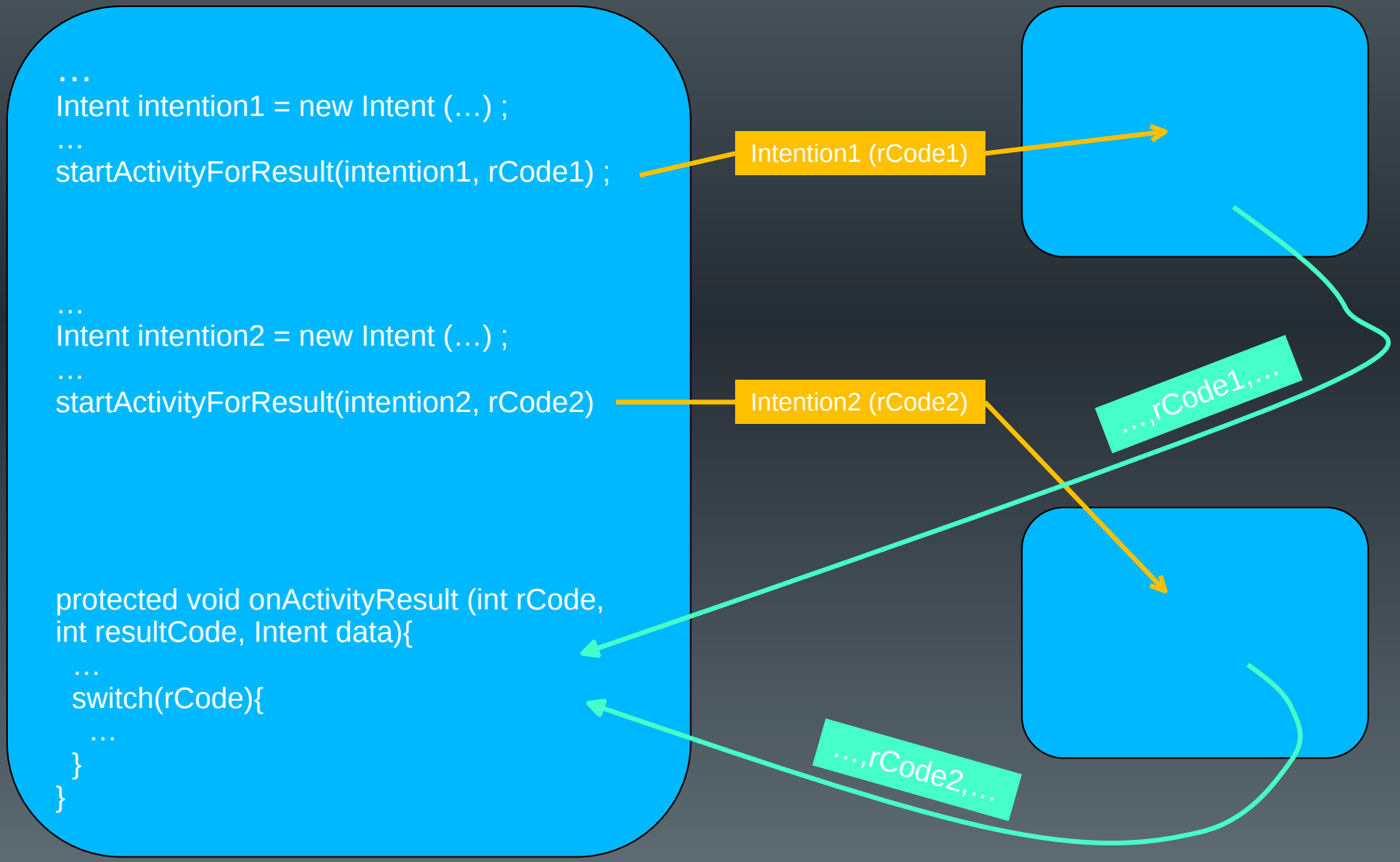
Code créé par le développeur pour identifier de manière unique son intention (>0)

```
void maFonction(...) {
  ...
  Intent intention = new Intent (…) ;
  …
  startActivityForResult(intention, requestCode) ;
  …
}


protected void onActivityResult (int requestCode, int resultCode, Intent data){
  …
}
```

Code de retour d'exécution de l'intent
Activity.RESULT_OK
Activity.RESULT_CANCELED

Données transmises en retour à l'activité appelante

Intentions avec résultats (émetteur)



Intentions avec résultats (récepteur)

```
void maFonction(...) {
  ...
  Intent intention = new Intent (…) ;
  …
  startActivityForResult(intention,
                         requestCode) ;
  …
}


protected void onActivityResult (int requestCode,
                                 int resultCode,
                                 Intent data){
  …
}
```

```
public void onCreate (Bundle savedInstanceState)
{
    // récupérer l'intent
    Intent intention = getIntent () ;

    // extraire les données et les traiter
    …
    // créer l'intent résultat
    Intent resultat = new Intent();
    // ajout des résultats
    …
    // prépare le retour des résultats
    setResult(RESULT_OK, resultat);
    finish();
}
```

ou

```
setResult(RESULT_CANCELED, resultat) ;
finish();
```

Remarques (1)

(1)

```
void maFonction(...) {
  ...
  Intent intention = new Intent (…) ;
  …
  startActivityForResult(intention,
                         requestCode) ;
  …
}

protected void onActivityResult (int requestCode,
                                 int resultCode,
                                 Intent data){
  …
}
```

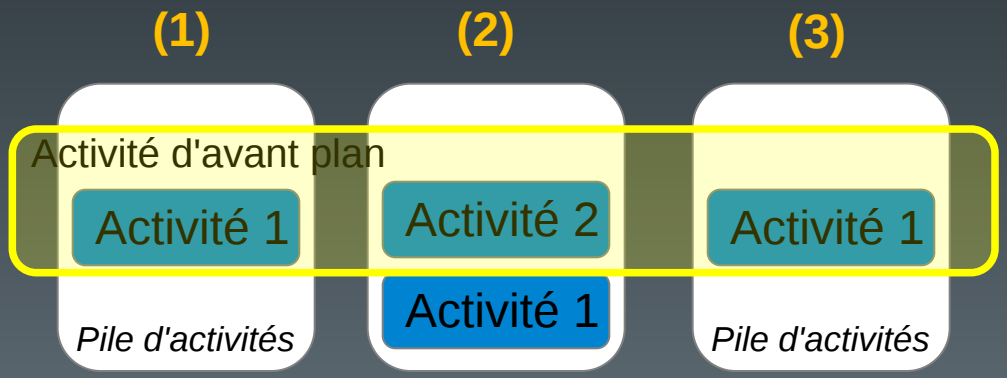
Activité1

(2)

(3)

```
public void onCreate (Bundle savedInstanceState)
{
   // récupérer l'intent
   Intent intention = getIntent () ;

   // extraire les données et les traiter
   …
   // créer l'intent résultat
   Intent resultat = new Intent();
   // ajout des résultats
   …
   // prépare le retour des résultats
   setResult(RESULT_OK, resultat);
   finish();
}
```

Activité2



Remarques (2)

(1)

```
public void onCreate (Bundle savedInstanceState) {
    // récupérer l'intent
    Intent intention = getIntent () ;
    …
}

void maFonction(...) {
  ...
  Intent intention = new Intent (Activité2) ;
  …
  startActivity(intention);
  …
}
```

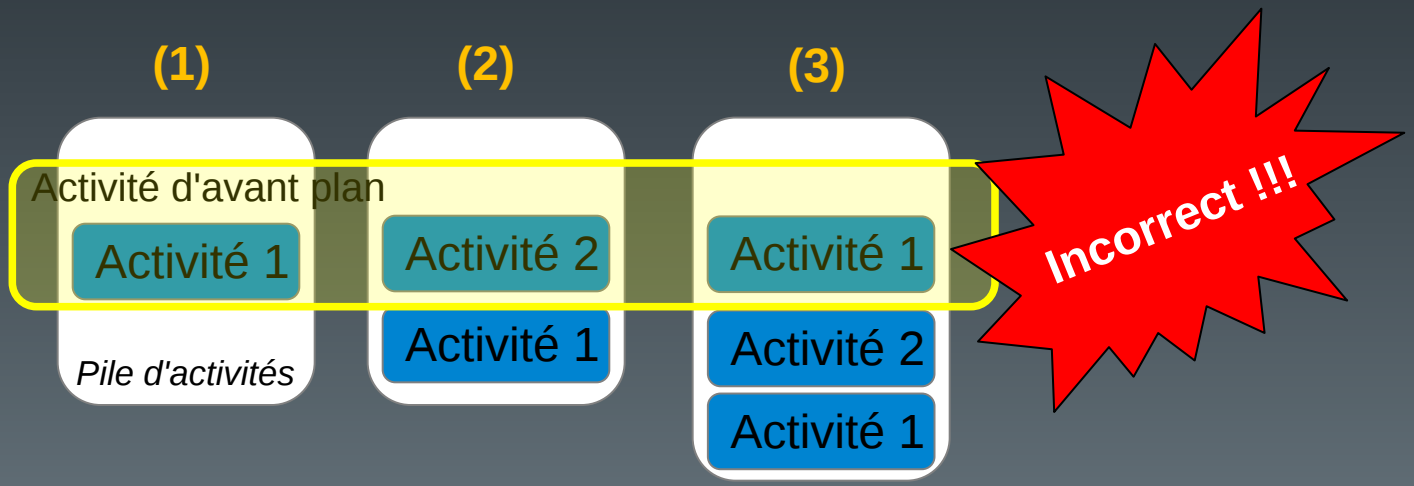
Activité1

(2)

(3)

```
public void onCreate (Bundle savedInstanceState)
{
  // récupérer l'intent
  Intent intention = getIntent () ;
  …

  Intent resultat = new Intent(Activité1);
  // ajout des résultats
  …
  // renvoyer les résultats
          startActivity(resultat);
}
```

Activité2



Intentions avec résultats (1)

- Modification du mécanisme à partir d'Android 12
  - Remplacement de startActivityForResult par registerForActivityResult
    - plus de onActivityResult()
    - Fonction *callback* définie au moment de la création de la classe de lancement
  - Quelques raisons
    - Résultats manqués si le composant de départ est recréé
    - Conflits avec le même code de requête
    - ...

Intentions avec résultats (2)

- Plus de surcharge de onActivityResult()
  - Lancement à partir d'une classe ActivityResultLauncher<>
    - Template spécifiant le type de résultat attendu
    - Méthode de lancement launch()

Ce que reçoit le lanceur

  - Cas des intentions explicites

```
public class MainActivity extends AppCompatActivity {
    // attributs de la classe
    private ActivityResultLauncher<Intent> mGetResult = …

    protected void onCreate(Bundle savedInstanceState) { … }

    public void onClick(View v){
      Intent mess = new Intent(…);
      …
      mGetResult.launch(mess);
}
```

Intentions avec résultats (3)

- Création/enregistrement de la fonction callback
  - Classe registerForActivityResult()
    - Deux paramètres au constructeur :
      - Type d'activité (classe de contrat)
        - Type de données en entrée
        - Type de données en sortie
        - Nombreuses classes disponibles
      - La fonction callback à utiliser
    - Enregistre la fonction callback
    - Retourne un lanceur (ActivityResultLauncher)
      - Un lanceur par activité "fille"
        - Pas de requestCode

Intentions avec résultats (4)

- Cas des intentions explicites

```
public class MainActivity extends AppCompatActivity {
    // attributs de la classe
    private ActivityResultLauncher<Intent> mGetResult = registerForActivityResult(

            // classe de contrat pour une intention explicite
            new ActivityResultContracts.StartActivityForResult(),

            // callback pour une intention explicite
            new ActivityResultCallback<ActivityResult>() {
                @Override
                public void onActivityResult(ActivityResult result) {

                }// onActivityResult

            }// ActivityResultCallback

    );// registerForActivityResult


    protected void onCreate(Bundle savedInstanceState) {
      …
    }

  …
}
```

Ce que récupère la fonction callback

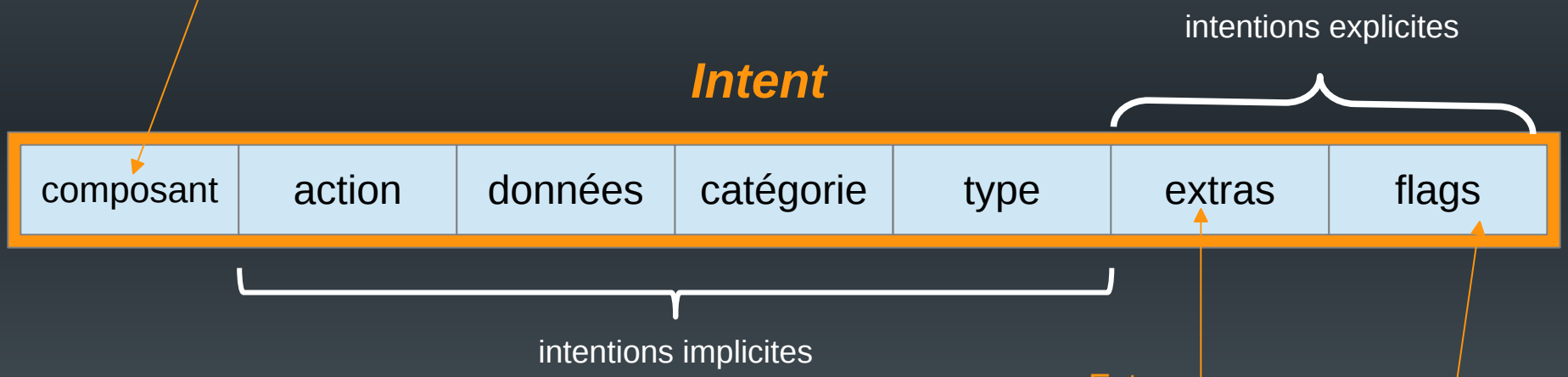
Intentions avec résultats (5)

- Cas des intentions explicites
  - Récupérer les résultats

```
private ActivityResultLauncher<Intent> mGetResult = registerForActivityResult(

        // classe de contrat pour une intention explicite
        new ActivityResultContracts.StartActivityForResult(),

        // callback pour une intention explicite
        new ActivityResultCallback<ActivityResult>() {
            @Override
            public void onActivityResult(ActivityResult result) {

                if(result.getResultCode() == RESULT_OK){
                  Intent mess = result.getData();
                  …
                }

            }// onActivityResult

        }// ActivityResultCallback

);// registerForActivityResult
```

Structure générale des Intentions

Nom du composant de destination
Optionnel : utilisé uniquement pour les intentions explicites



Extras
contenu supplémentaire transitant entre les composants

Indicateurs
permettant de modifier le comportement du cycle de vie des activités

Spécifier le nom du composant

- Différentes méthodes utilisables :
  - Intent(...) : le constructeur
  - SetComponent(...), setClassName(...), setClass(...), etc.

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

La classe

Le contexte

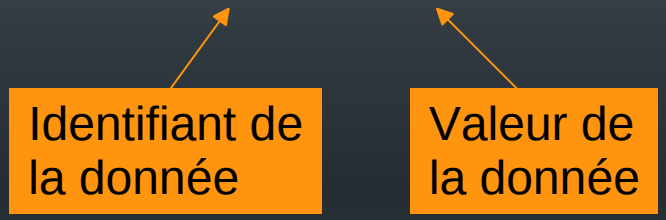
Le package

```
…
Intent = new Intent() ;
Intent.setClassName(this, "fr.univ-littoral.renaud.bidon.SecondActivity") ;
…
startActivity(intent) ;
...
```

```
…
Intent = new Intent() ;
Intent.setComponent(new ComponentName("fr.univ-littoral.renaud.bidon",
                                      "fr.univ-littoral.renaud.bidon.SecondActivity") ;
…
startActivity(intent) ;
...
```

Les extras (1)

- Permettent d'insérer des données dans l'intention
  - Communications inter-composants
  - Structure : (clé, valeur)



type = n'importe quel type de base (boolean, int, String, float[], ...)

insertion

Intent.putExtra(String cle, *type* valeur) ;

- Clé définie par l'utilisateur

public final static String MACLE = "fr.ulco.renaud.appli.MACLE" ;

- Quelques clés prédéfinies

récupération

type get{*type*}Extra(String cle, *type* vdefaut) ;

- vdefaut : valeur par défaut renvoyée si la clé n'est pas trouvée dans l'intention
- pas de valeur par défaut pour les tableaux, ni les String

type[] get{Type}ArrayExtra(String cle) ;
String getStringExtra(String cle) ;

Les extras (2)

```
public class mainActivity extends Activity {

  public final static String MARQUE = "fr.ulco.renaud.appli.MARQUE" ;
  public final static String PUISS = "fr.ulco.renaud.appli.PUISS" ;


...
  Intent intention = new Intent(this, otherActivity.class) ;
  String marque = "Citroen" ;
  int puissance = 6 ;
  intention.putExtra(MARQUE, marque) ;
  intention.putExtra(PUISS, puissance) ;
  startActivity(intention) ;
...
}
```

```
public class otherActivity extends Activity{
...
  Intent intention = getIntent() ;
  int p = intention.getIntExtra(mainActivity.PUISSANCE, 0) ;
  String m = intention.getStringExtra(mainActivity.MARQUE) ;
...
}
```

Les extras (3)

- Possibilité d'ajouter un Bundle :
  - Intent.putExtras(String cle, Bundle valeur) ;
  - Bundle getBundleExtras(String cle) ;
- Possibilité d'ajouter une intention :
  - Intent.putExtras(Intent valeur) ;
  - Recopie tous les extras de « valeur » dans l'intent appelant
- Possibilité d'ajouter des objets complexes :
  - Doivent être sérialisables
    - La classe d'origine doit implémenter l'interface Parcelable
    - Intent.putExtra(String cle, Classe valeur) ;
    - Classe getParcelableExtra(String cle) ;