

Introduction à la programmation sous Android

Christophe Renaud

M2 Informatique
Année 2023-2024

Version 6.5.0 du 23/10/2023

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)

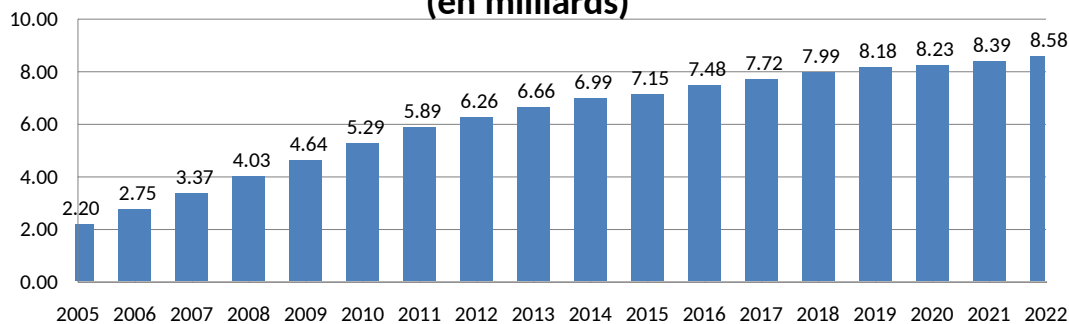
- Pourquoi développer des applications mobiles ?

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>

Nombre d'abonnés au mobile dans le monde
(en milliards)



<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

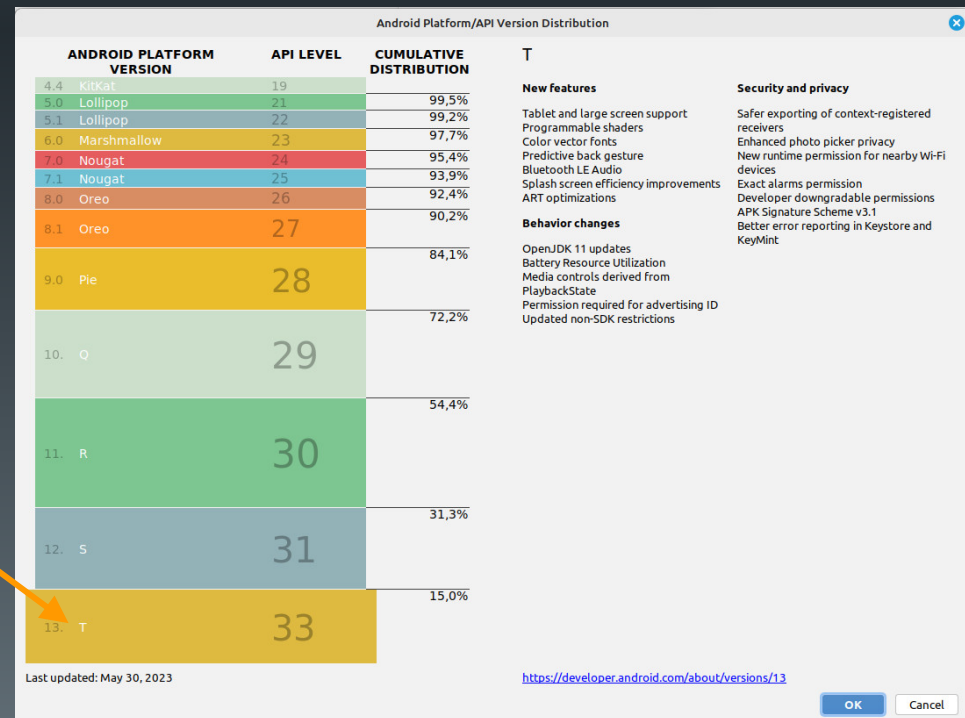
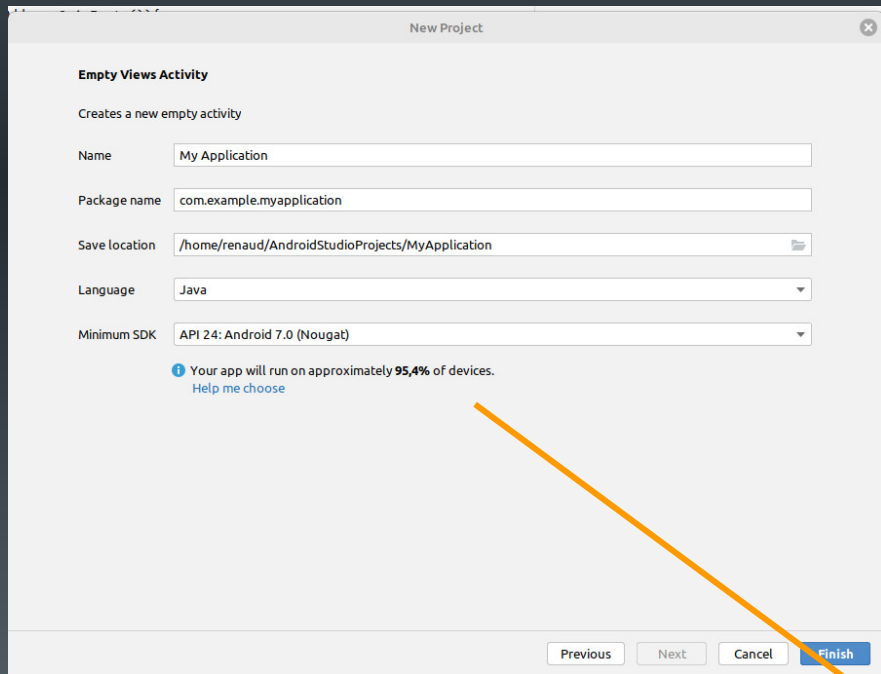
Android (3)

- 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 14.0 en 10/2023.
 - 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
14.0	U(Upside Down Cake)	34

Android (4)

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



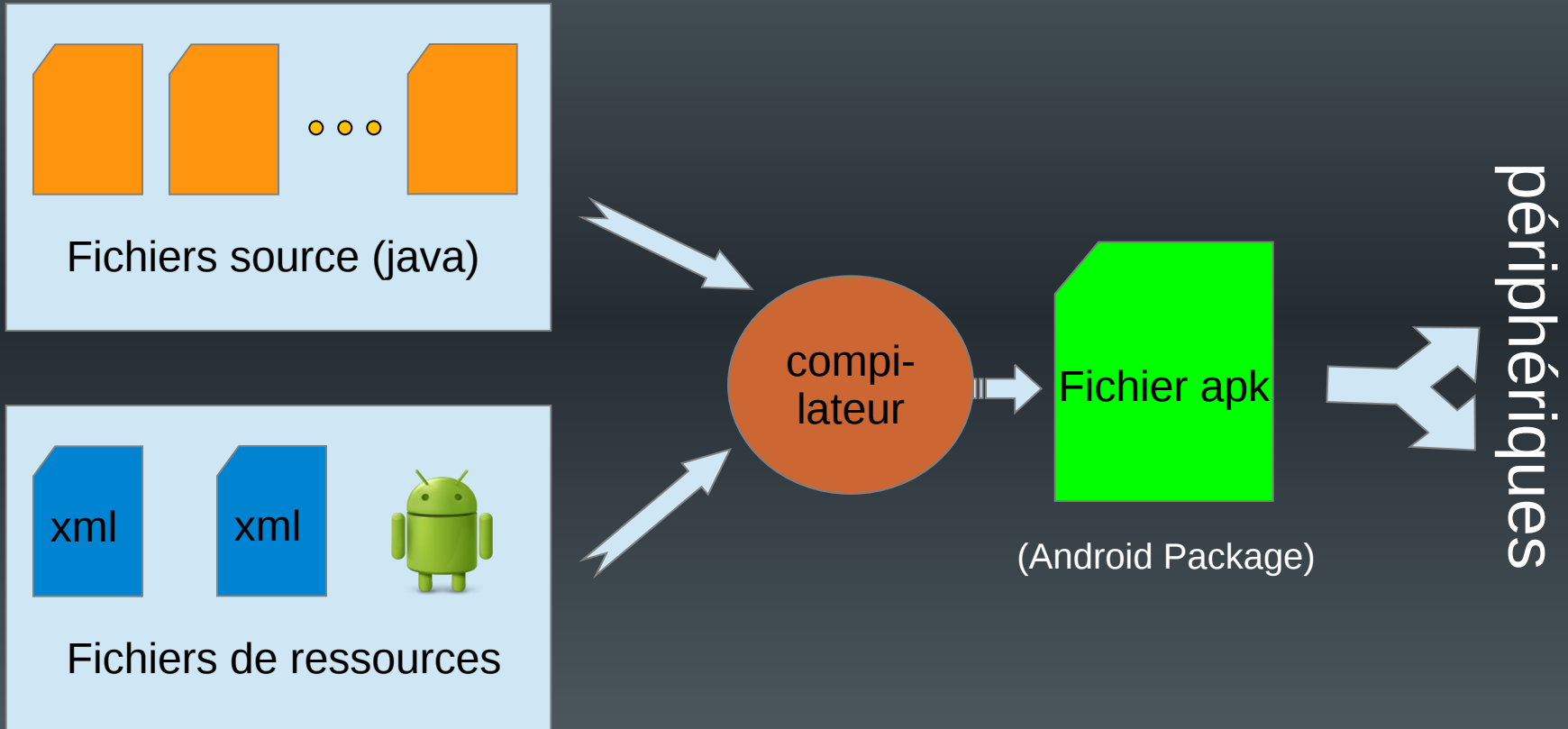
Android (5)

- 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

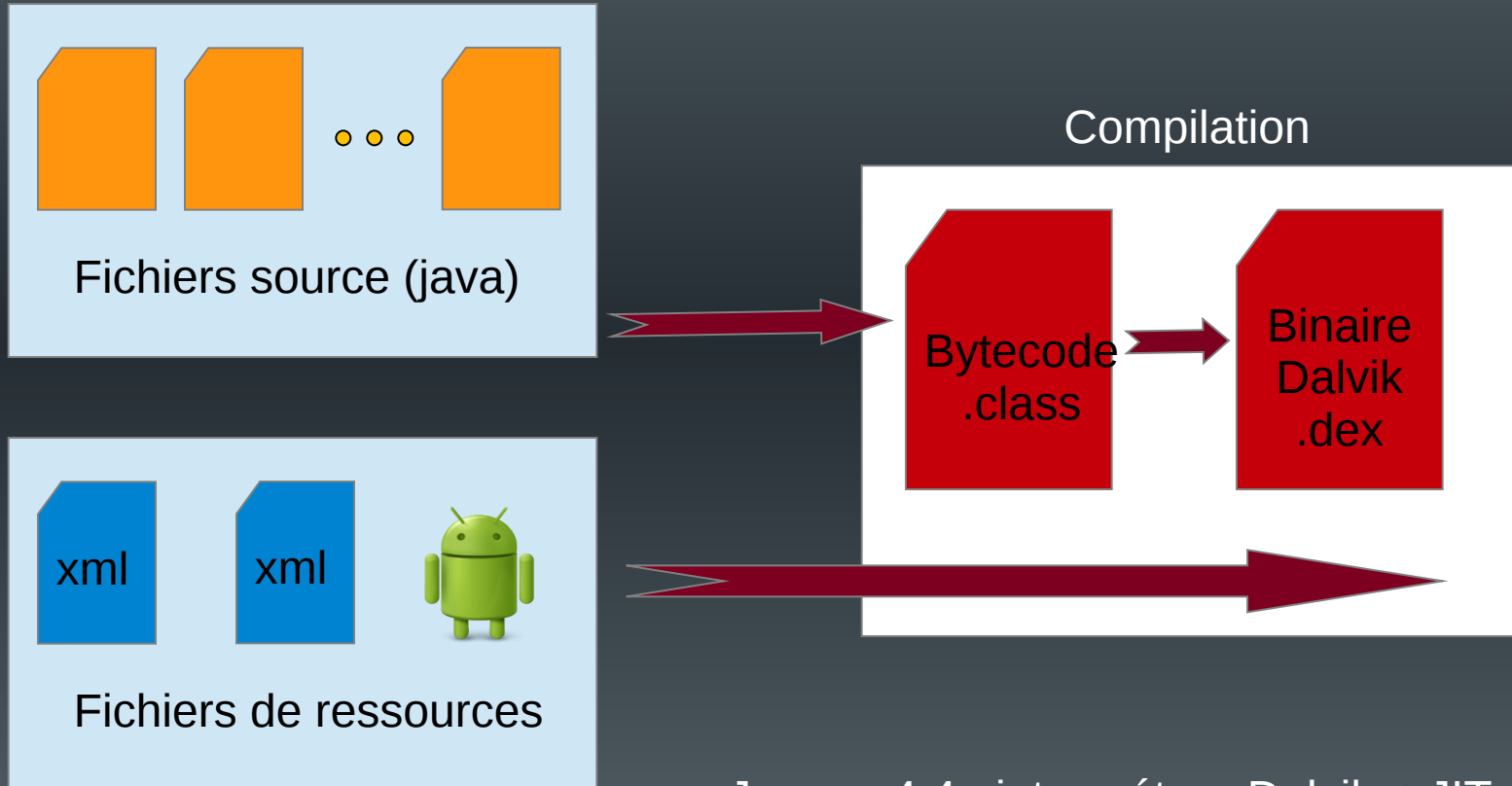
Plan du cours

- 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

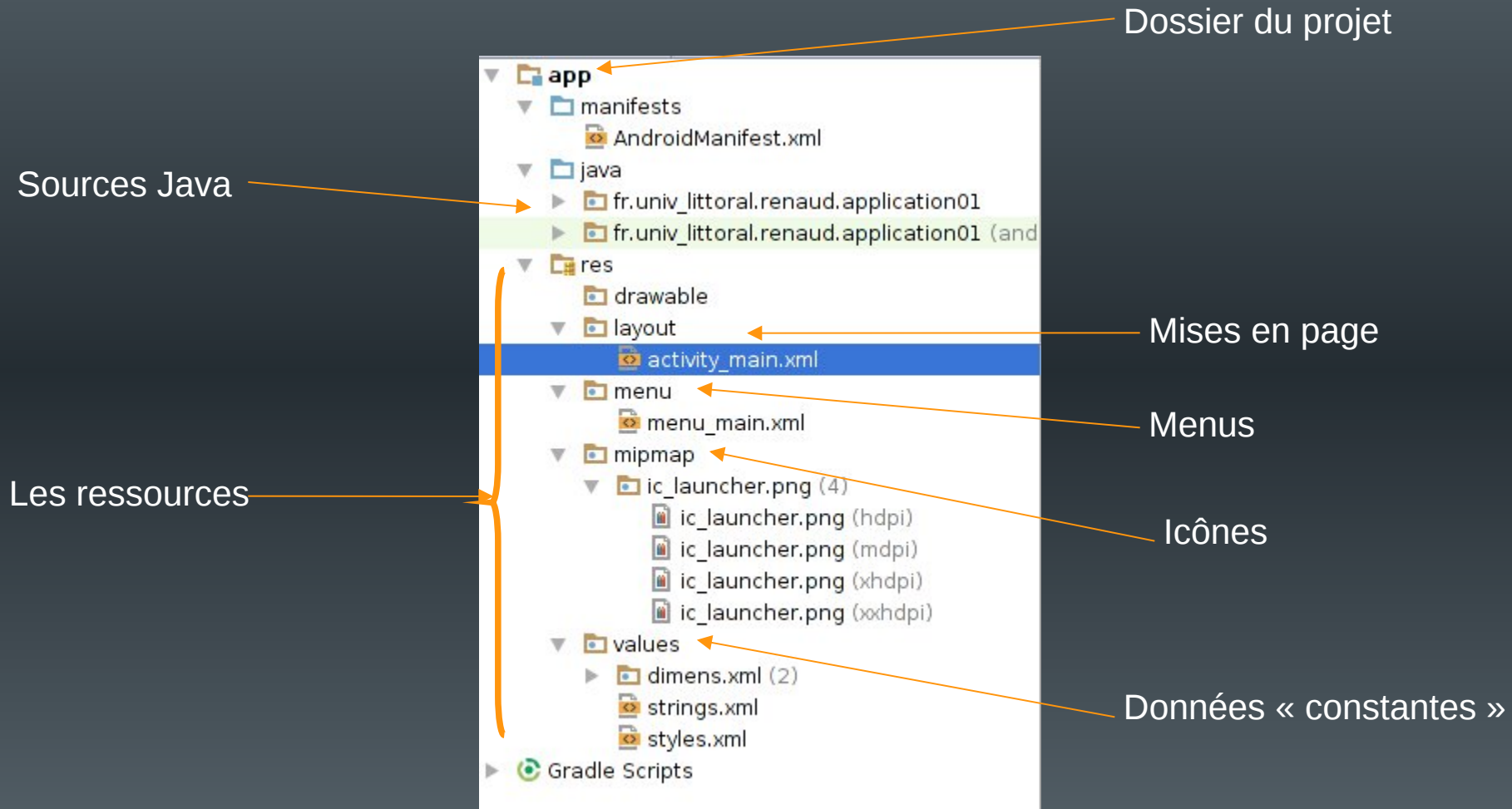


La compilation



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

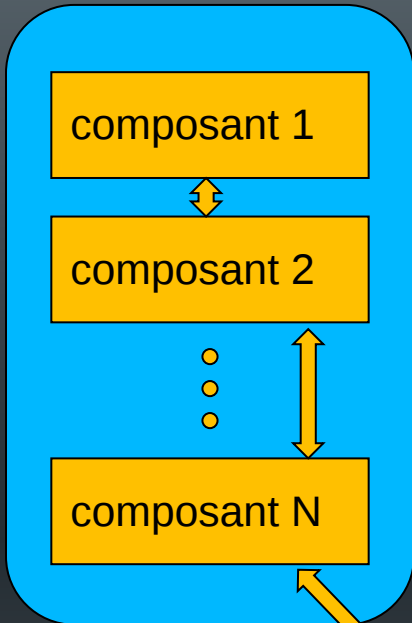


(sous Android Studio)

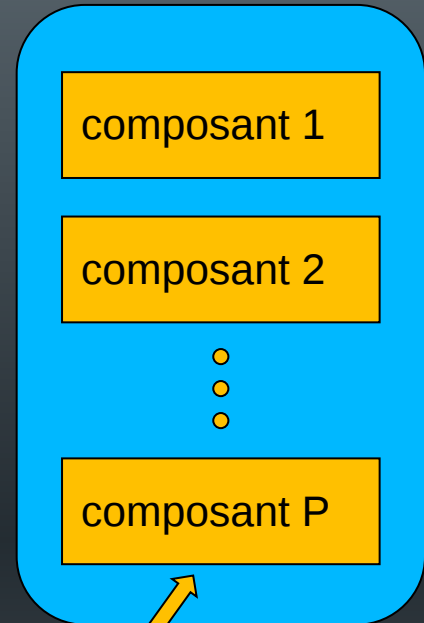
Les éléments d'une application

- 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 »

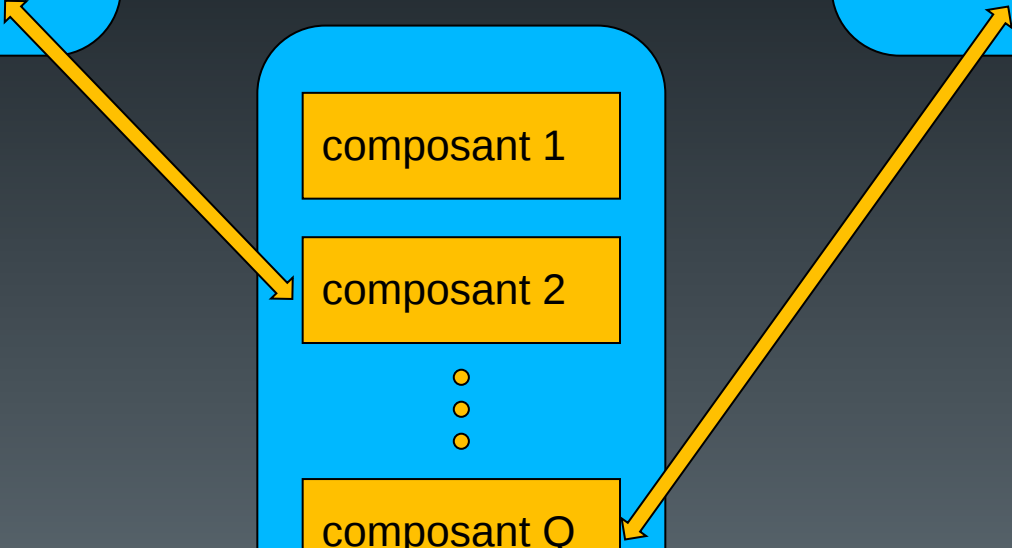
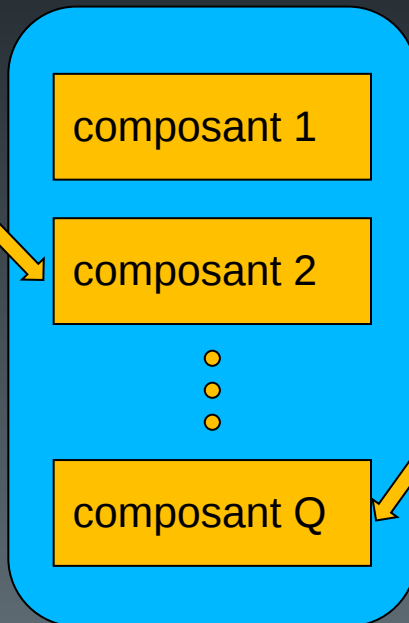
MonApplication



Application Y



Application X



Exemple

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



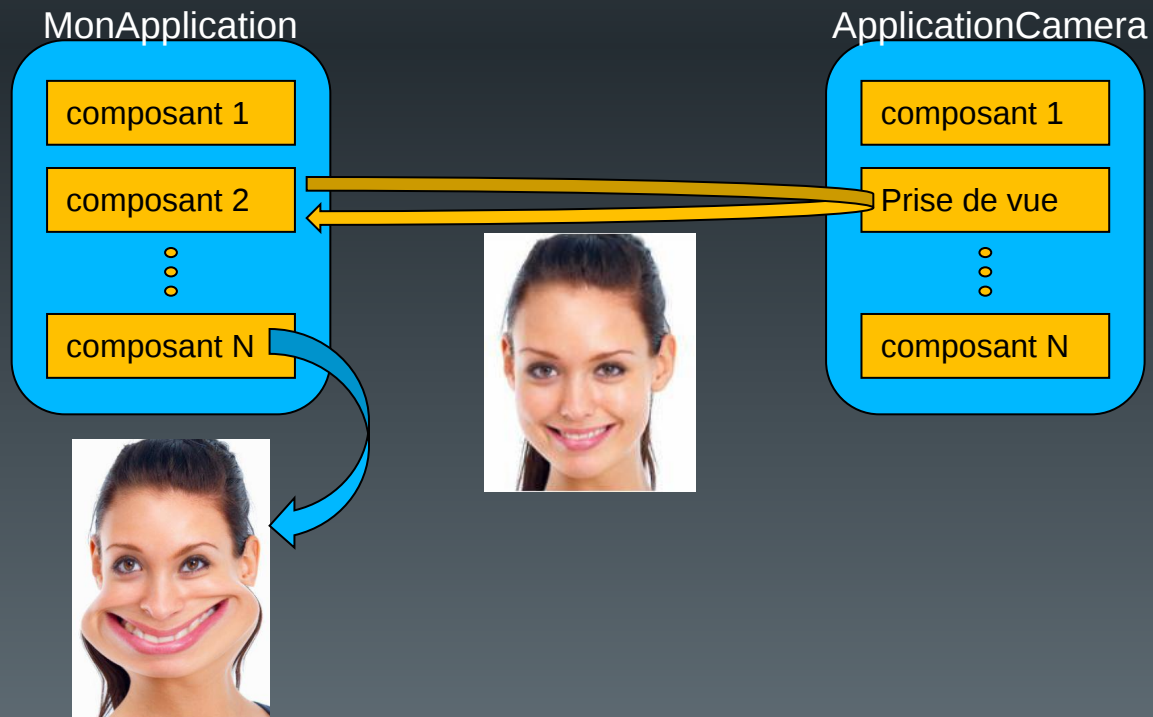
Ma spécialité : traitement d'images

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

Source : google play - effets du visage

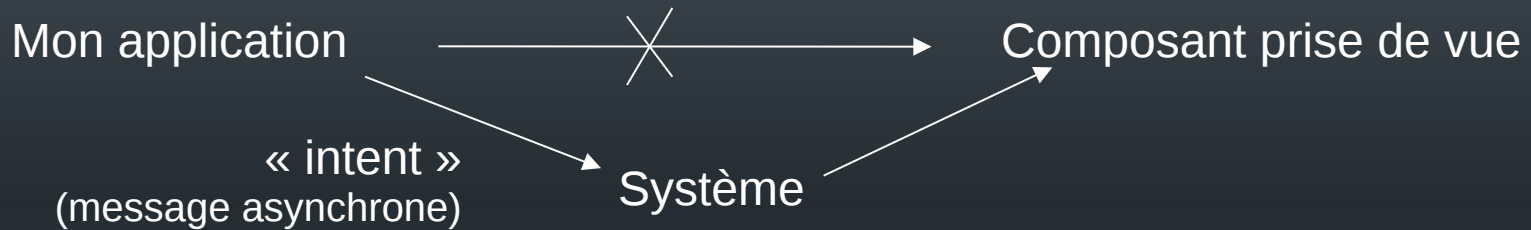
Exemple

- 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

Type de la constante

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

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

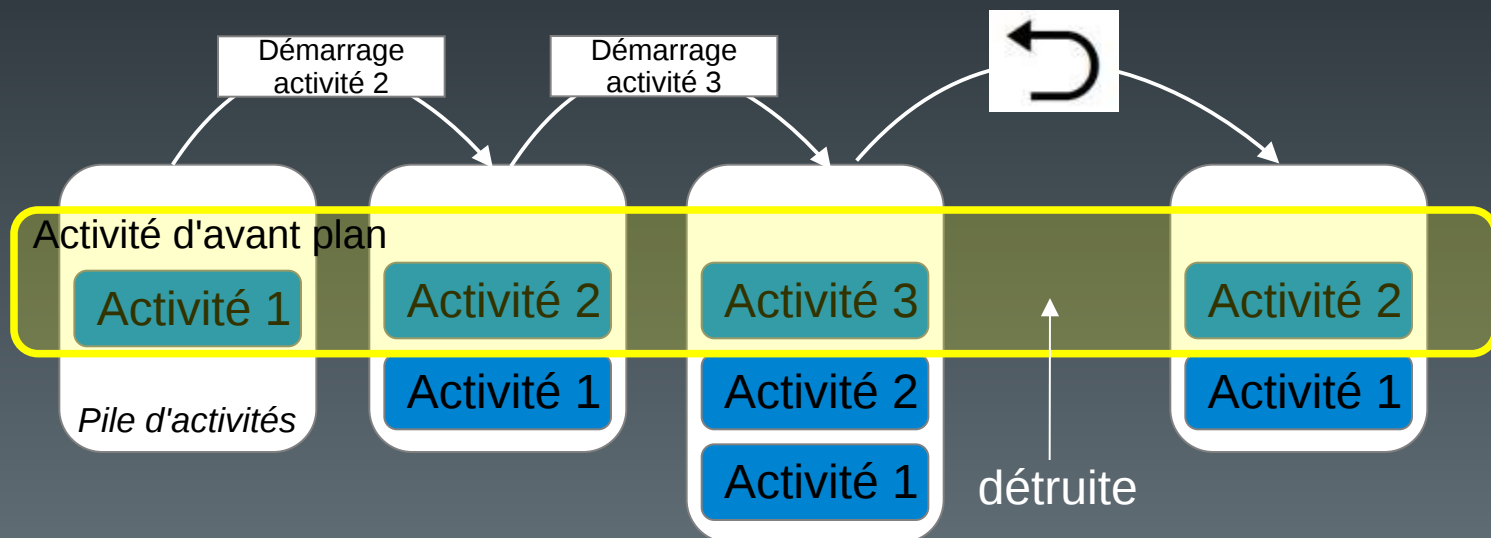
@string/action_settings

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

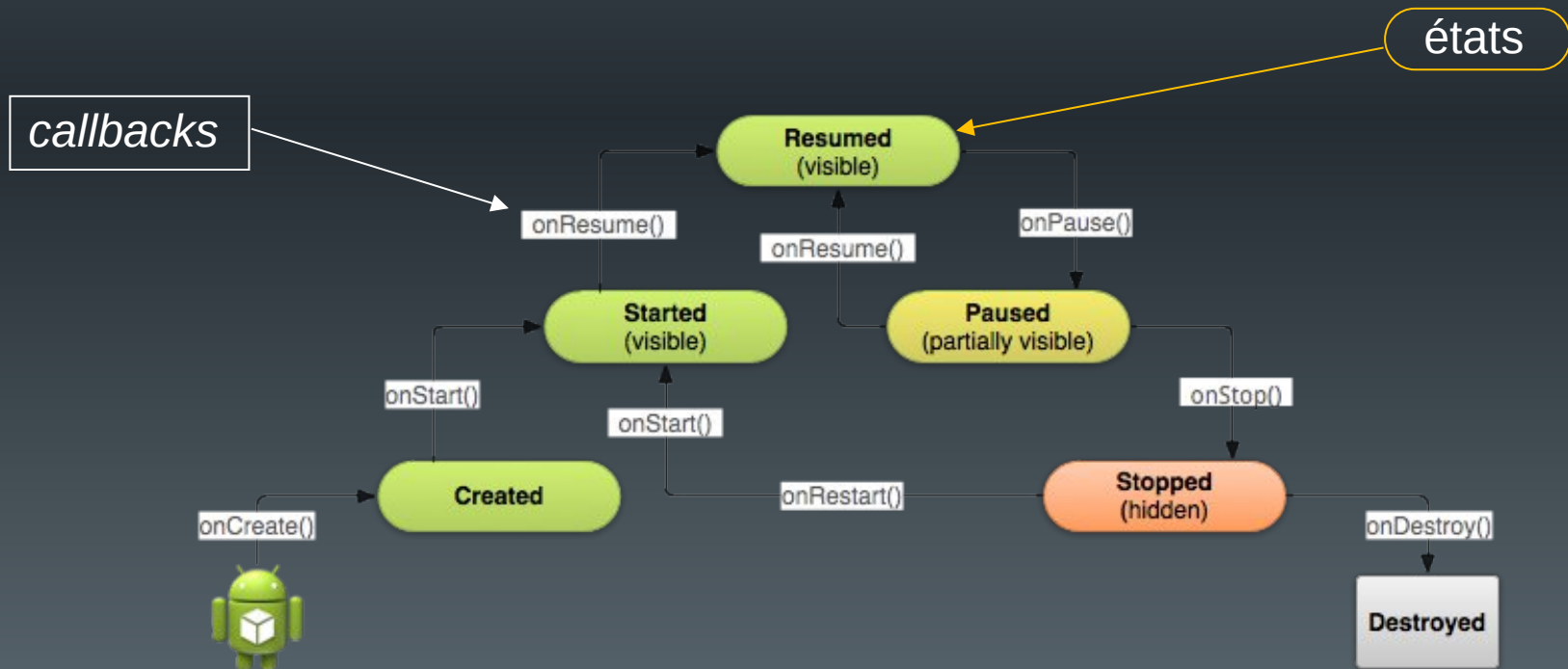
Les activités (1)

- 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 (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)

- Sauvegardes
 - `void onSaveInstanceState(Bundle outState)`

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

```
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);
```

} **Sauvegarde de la valeur des variables
internes dans le bundle**

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

```
}
```

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(PLOYER_NAME_KEY);
        puissance = savedInstanceState.getFloat(PLOYER_POWER_KEY);
    }
}
```

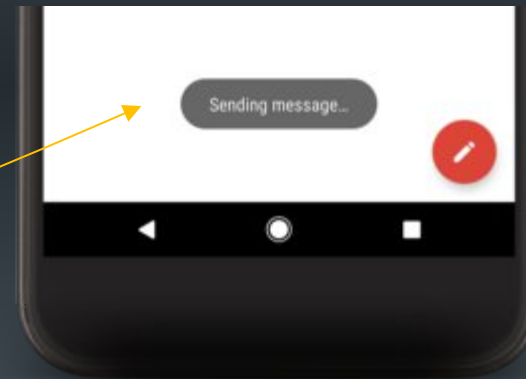
- **void onRestoreInstanceState (Bundle savedInstanceState)**

```
// 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(PLOYER_NAME_KEY);
    puissance = savedInstanceState.getFloat(PLOYER_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>

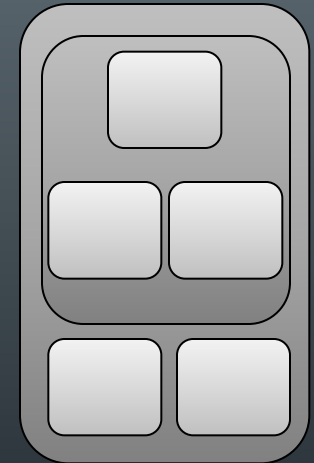
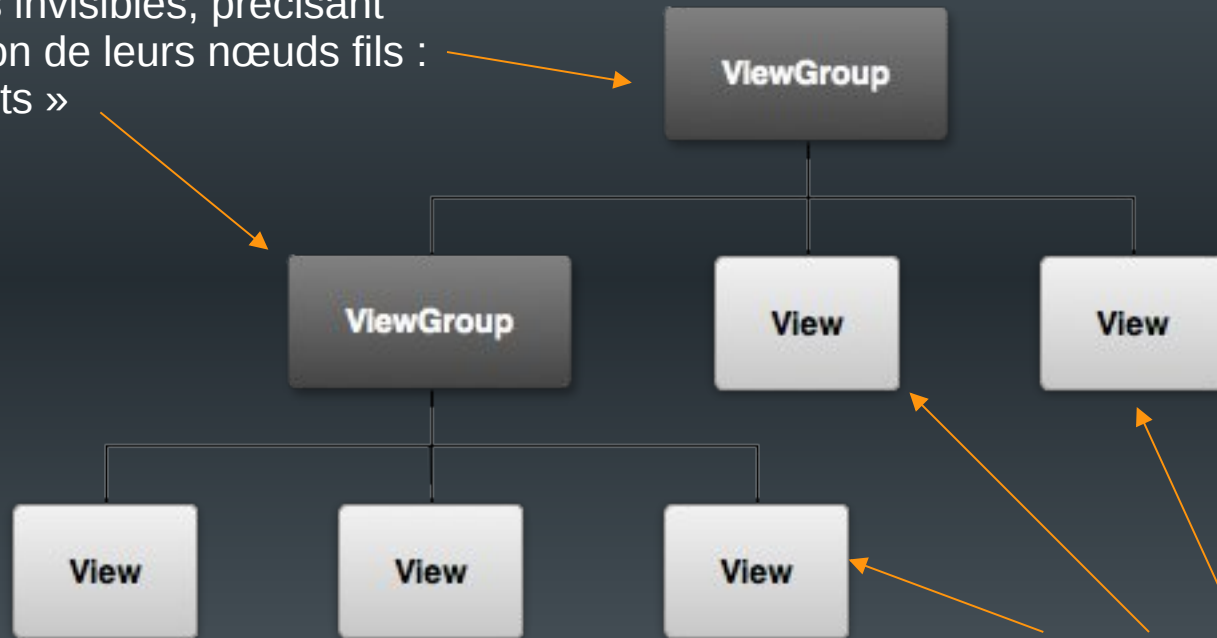
Définir une interface graphique

- 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 »



exemple

Source : developer.android.com

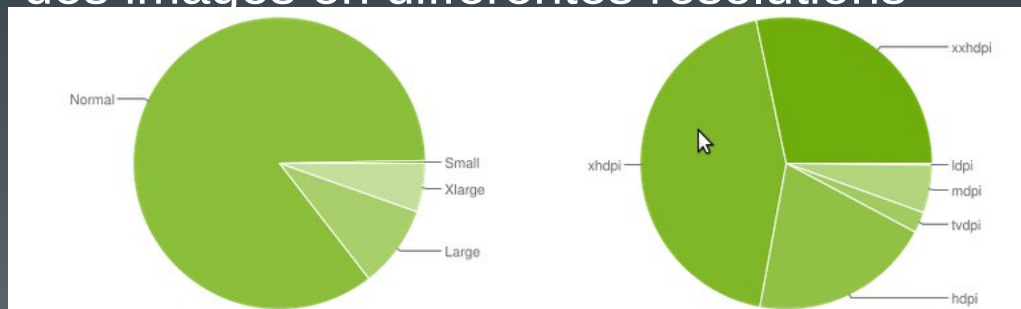
Objets graphiques permettant l'interaction (boutons, zones de texte, etc.) : les Widgets

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



Data collected during a 7-day period ending on July 13, 2020.
Any screen configurations with less than 0.1% distribution are not shown.

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



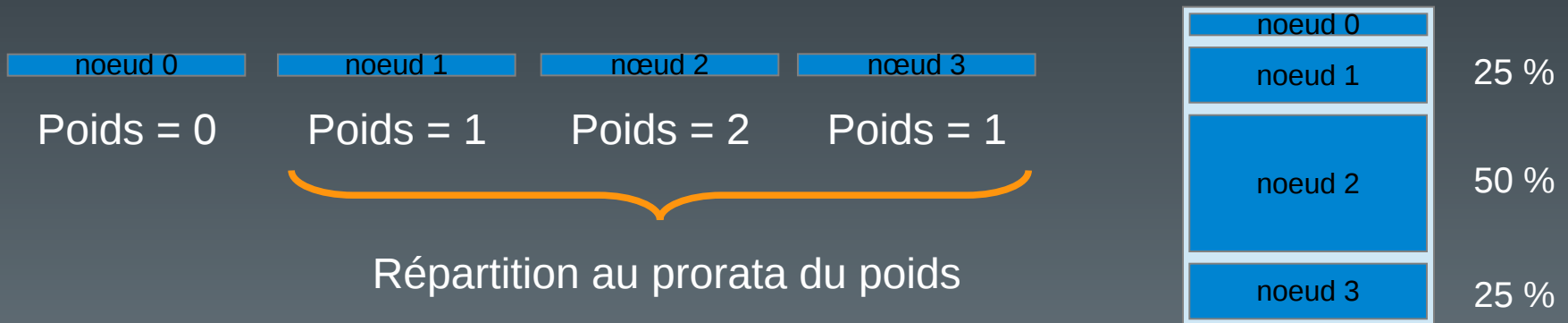
```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3   xmlns:app="http://schemas.android.com/apk/res-auto"
4   xmlns:tools="http://schemas.android.com/tools"
5   android:layout_width="match_parent"
6   android:layout_height="match_parent"
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"  
/>
```

40 %

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

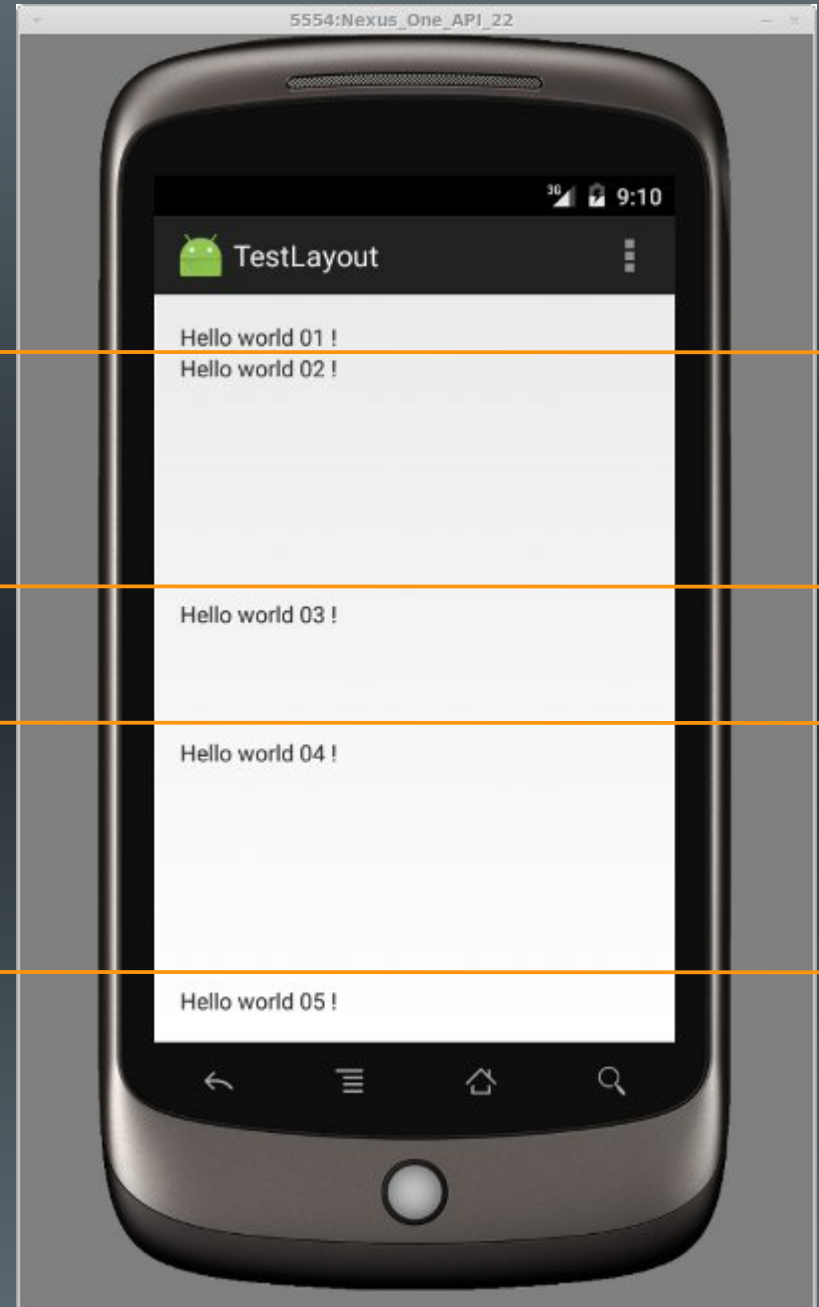
20 %

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

40 %

```
<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"  
/>
```



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

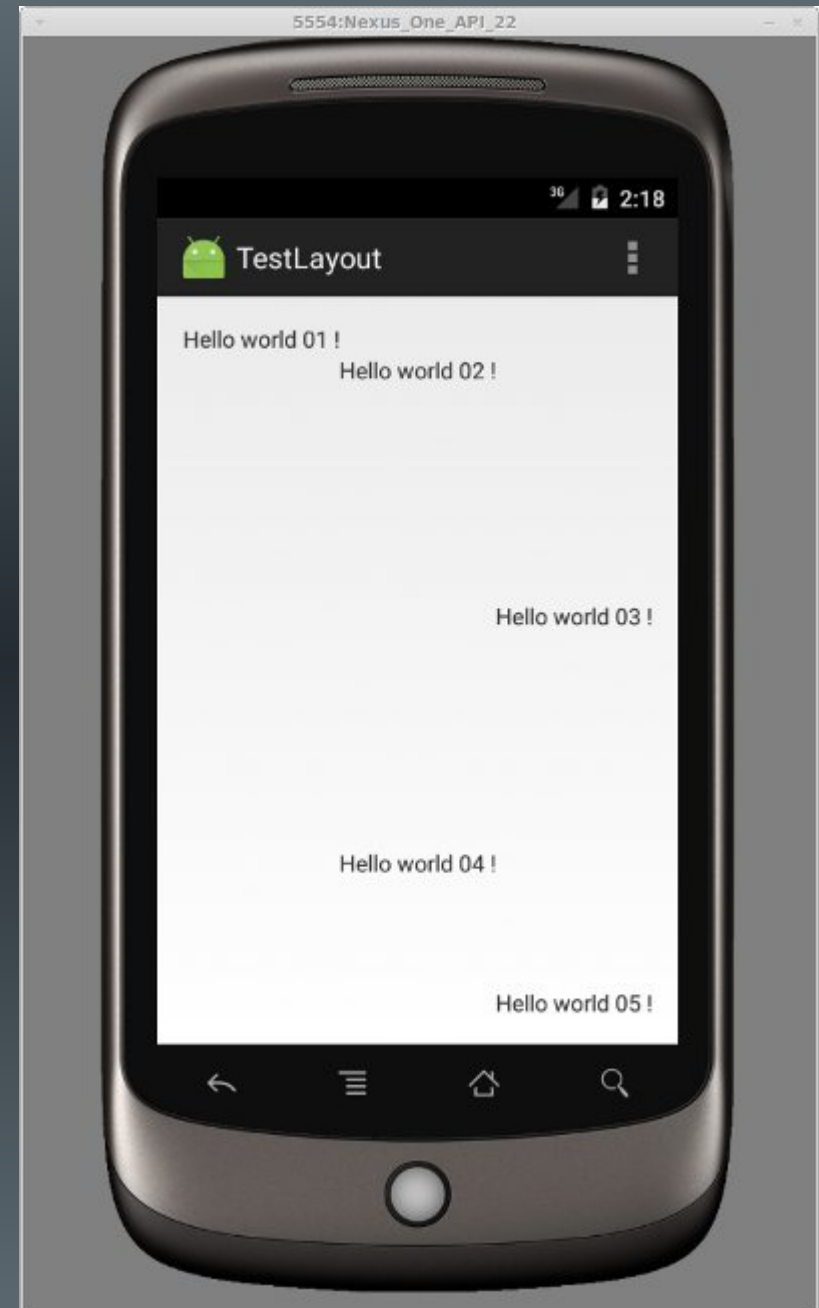
```
<TextView
    android:text="@string/hello_world_01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
/>

<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"
/>

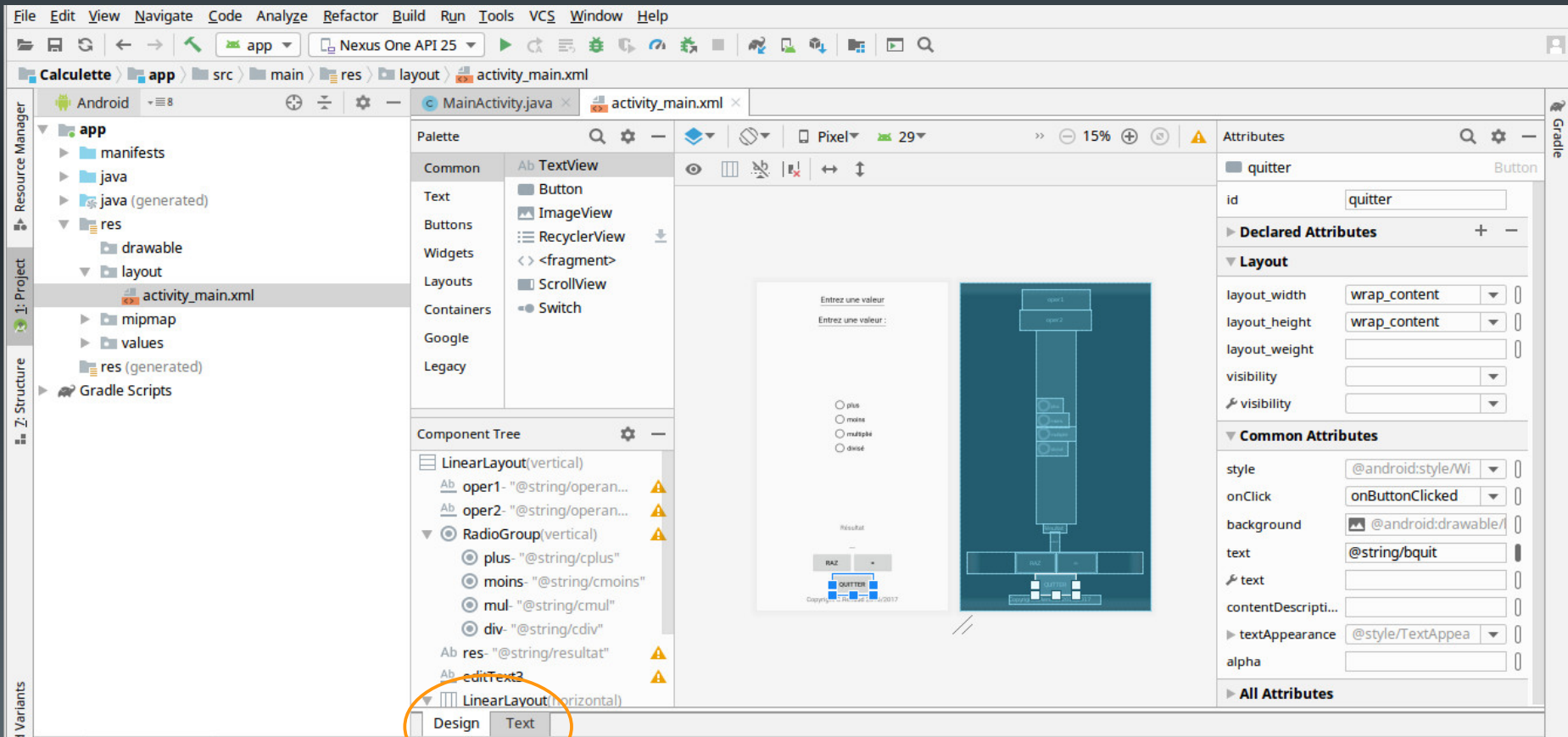
<TextView
    android:text="@string/hello_world_04"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:gravity="center"
    android:layout_weight="2"
/>

<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)

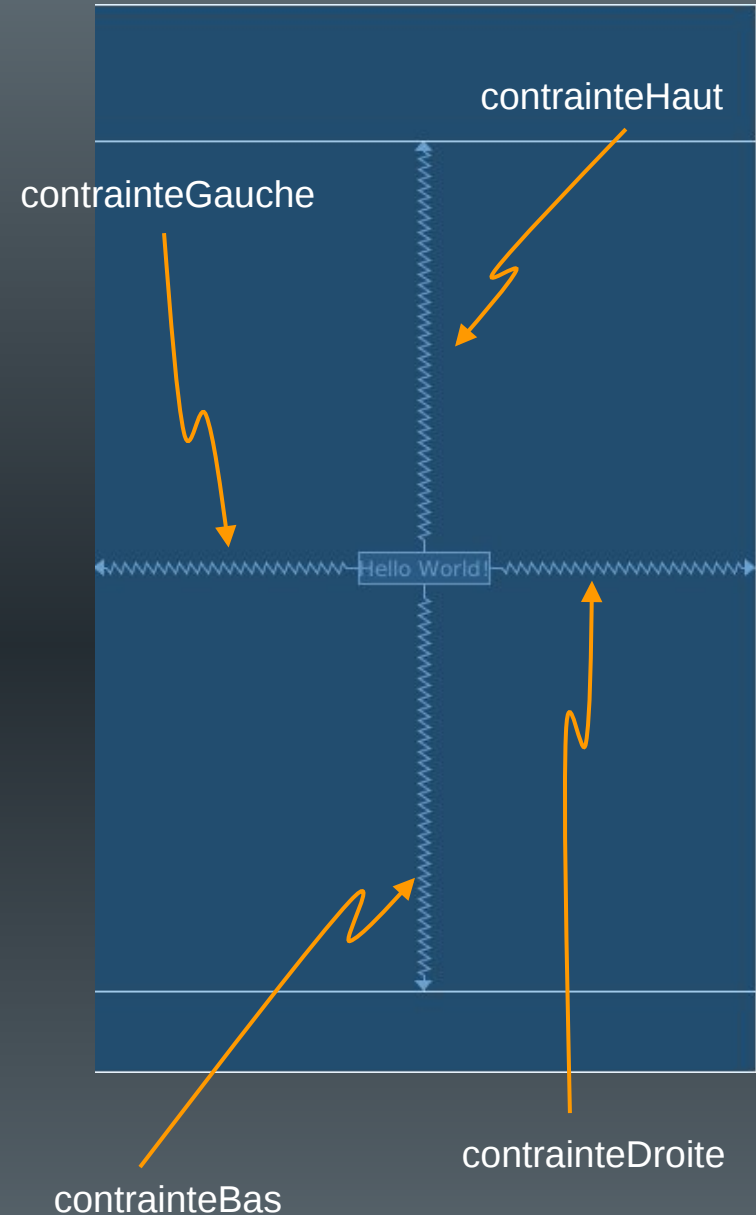
- Contraintes
 - Au moins une contrainte horizontale
 - Au moins une contrainte verticale

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>
```

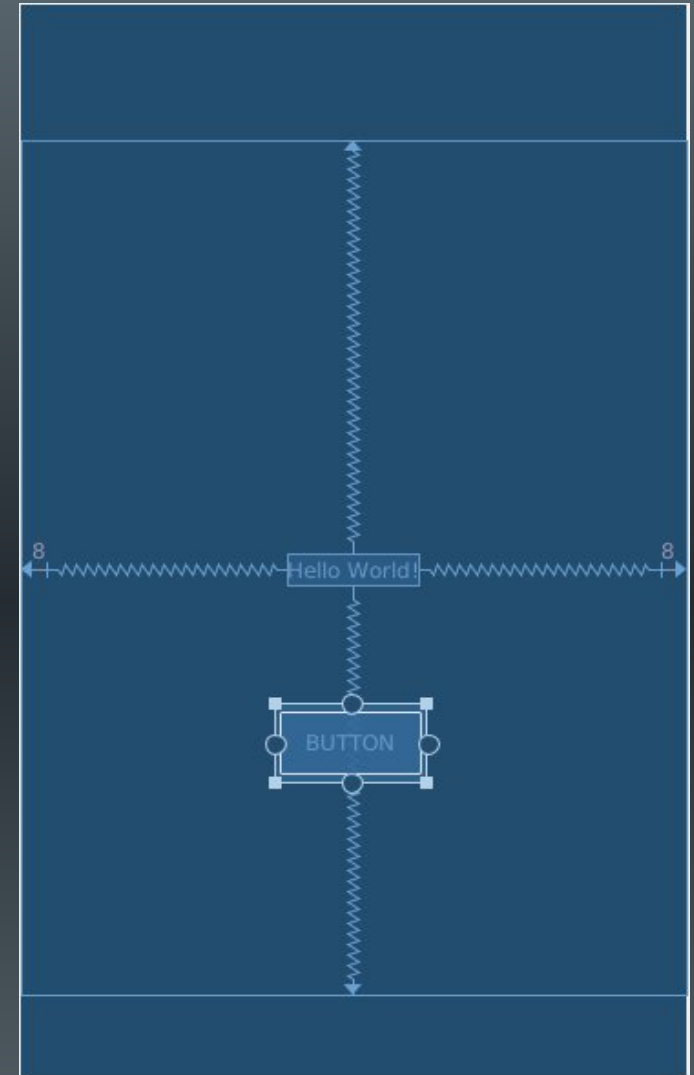
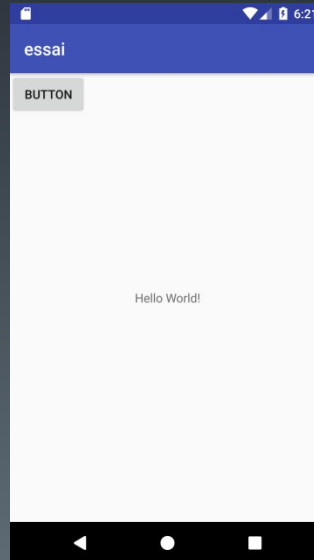


ConstraintLayout (2)

- 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

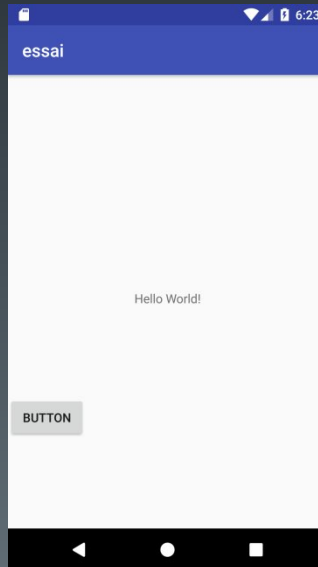
```
android:id="@+id/button"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:text="Button"  
tools:layout_editor_absoluteX="148dp"  
tools:layout_editor_absoluteY="373dp" />
```



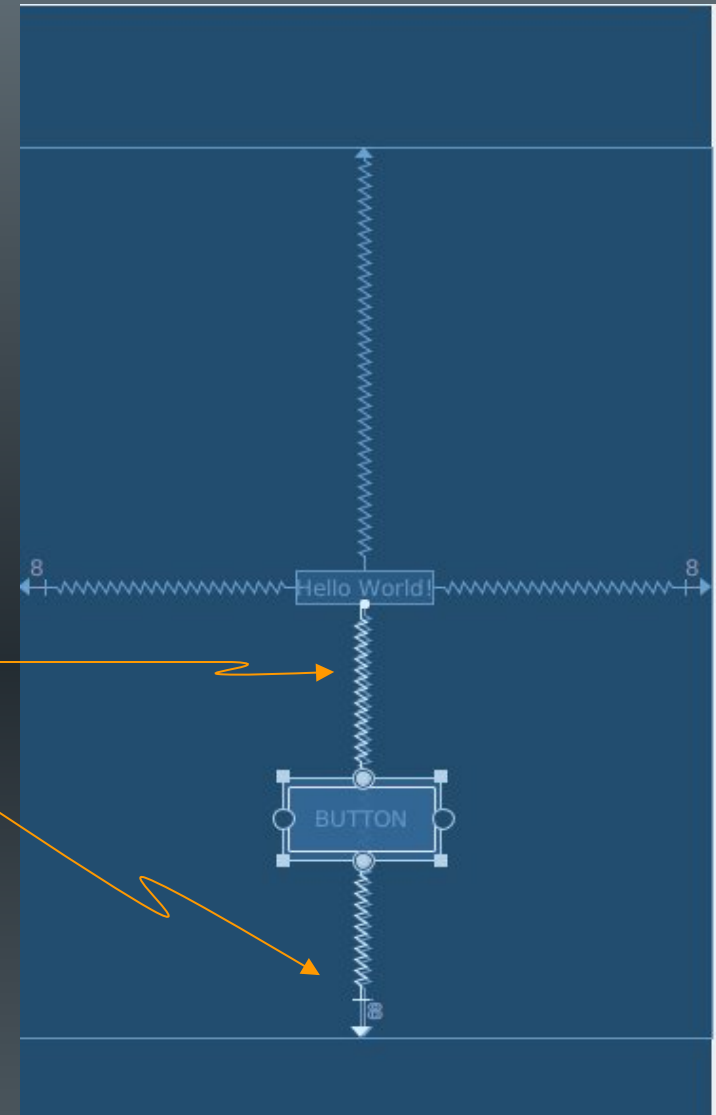
ConstraintLayout (3)

- 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" />
```



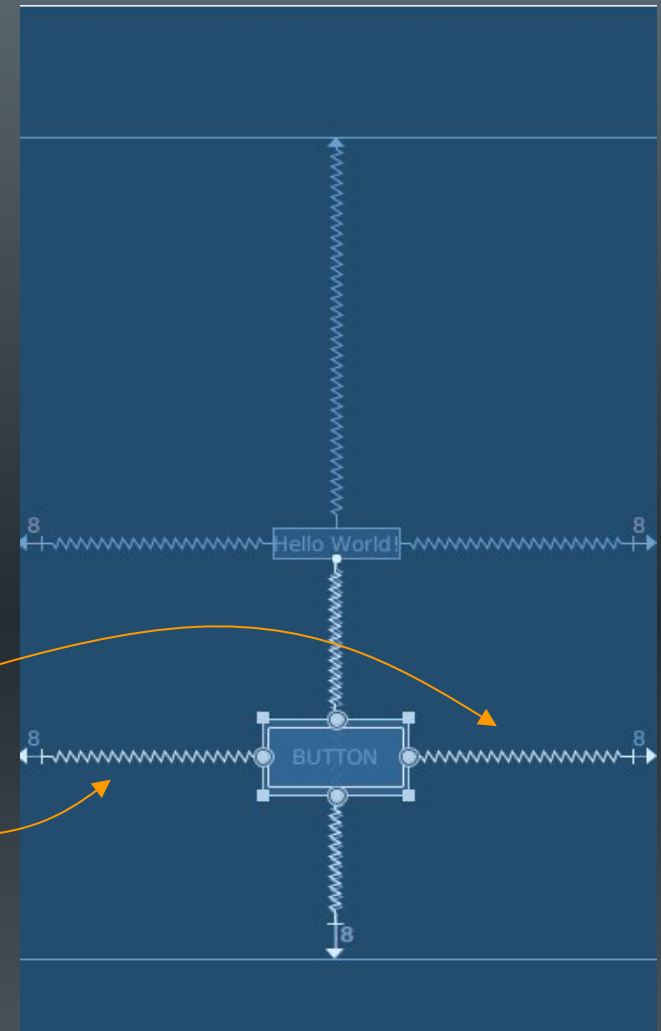
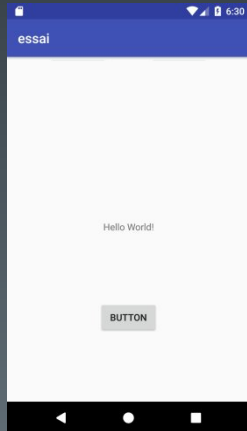
X=0



ConstraintLayout (4)

- Ajout d'une contrainte

```
<Button horizontale
    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" />
```

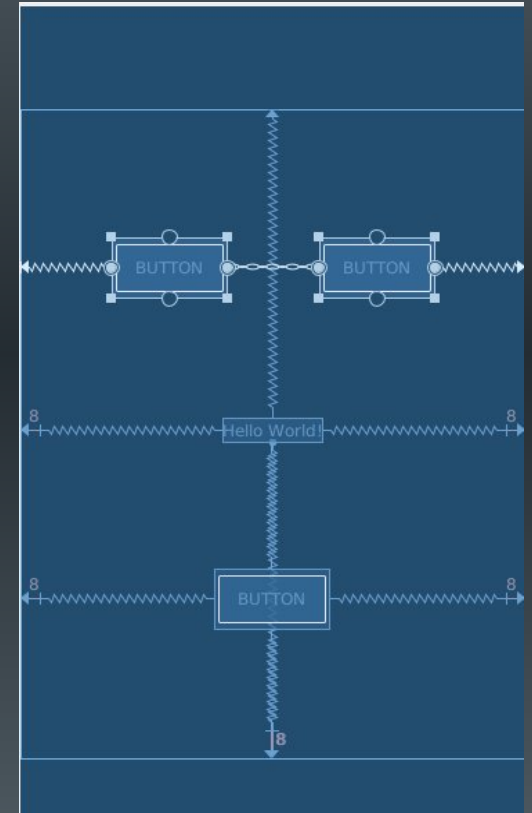
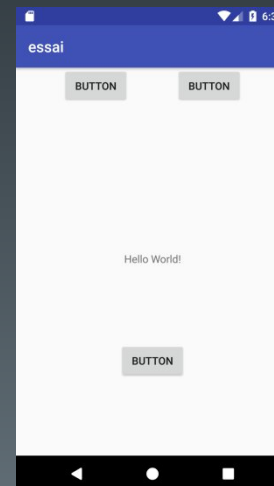
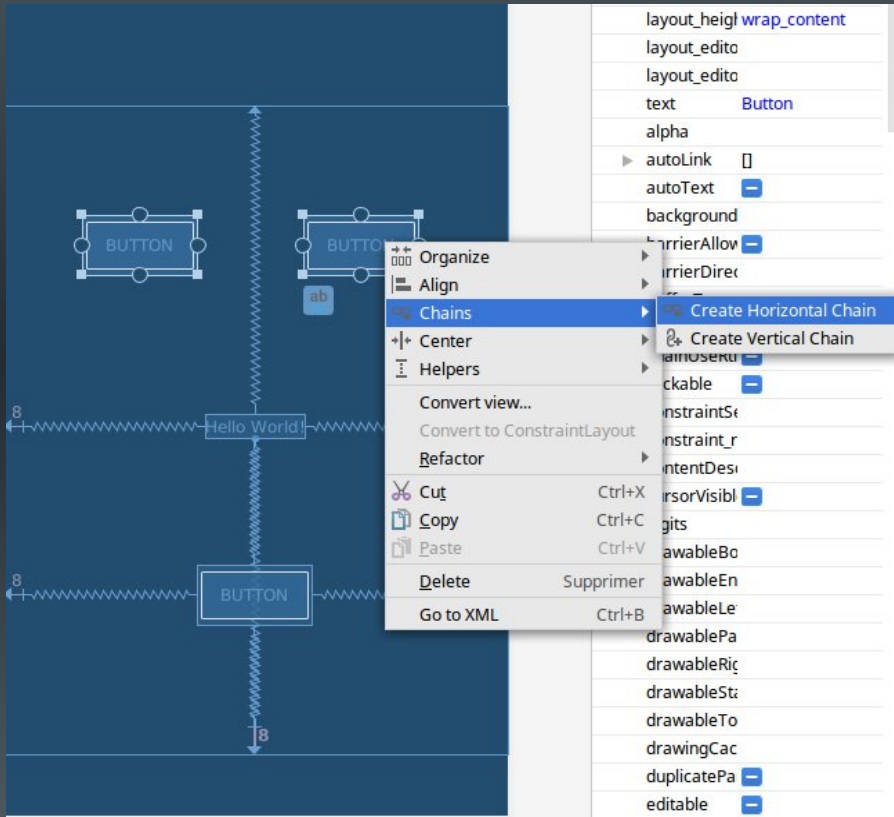


Remarque :

- @+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

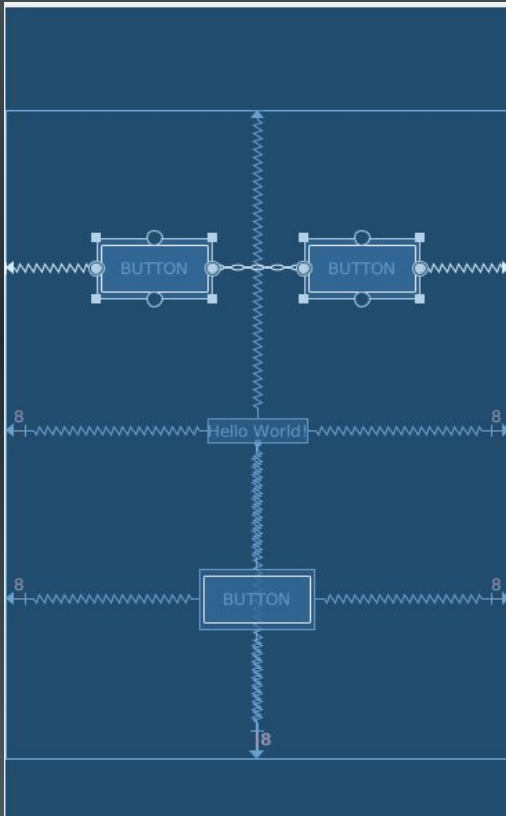
ConstraintLayout (5)

- Création de chaînages



Pas de contraintes verticales
(y = 0)

ConstraintLayout (6)



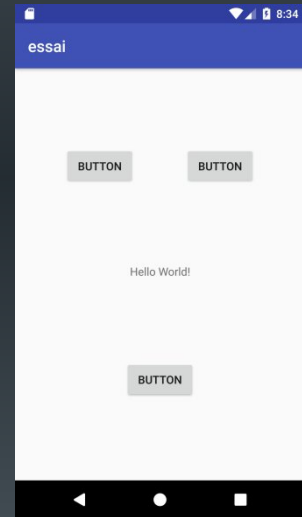
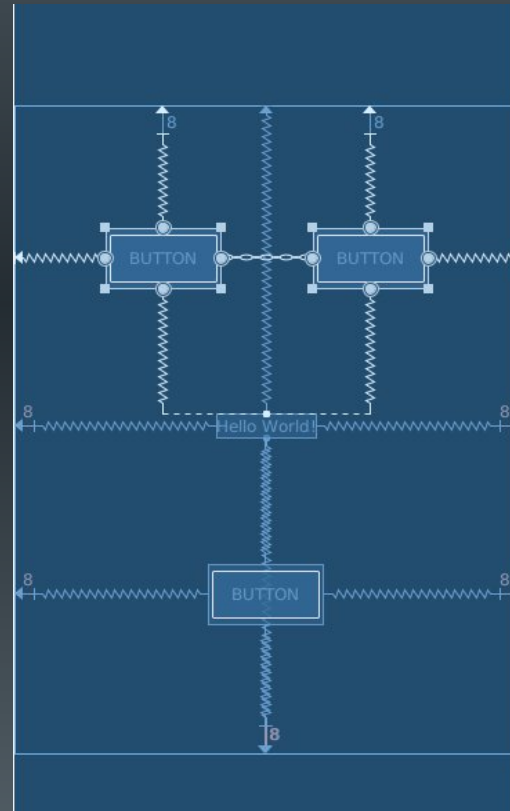
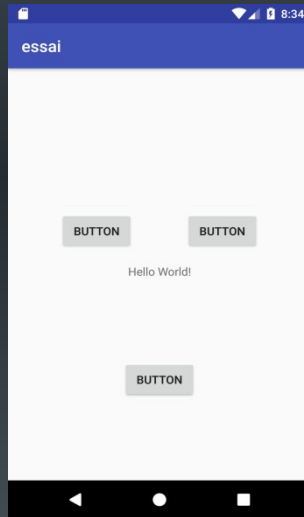
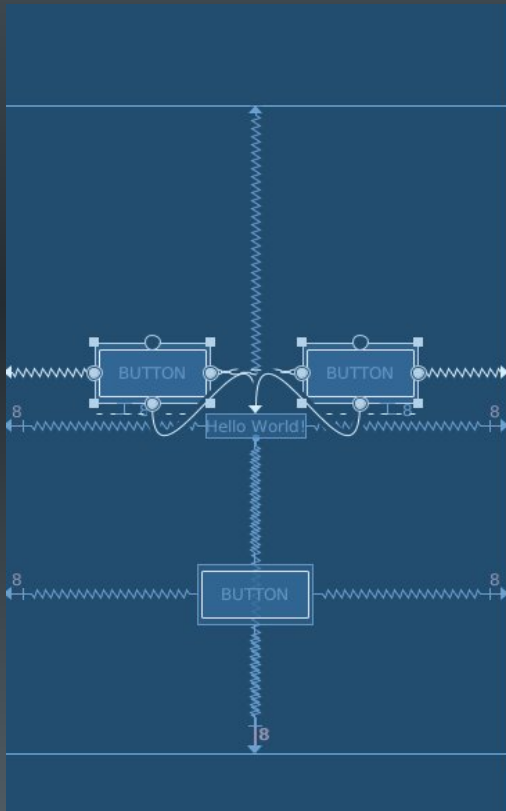
<Button

```
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" />
```

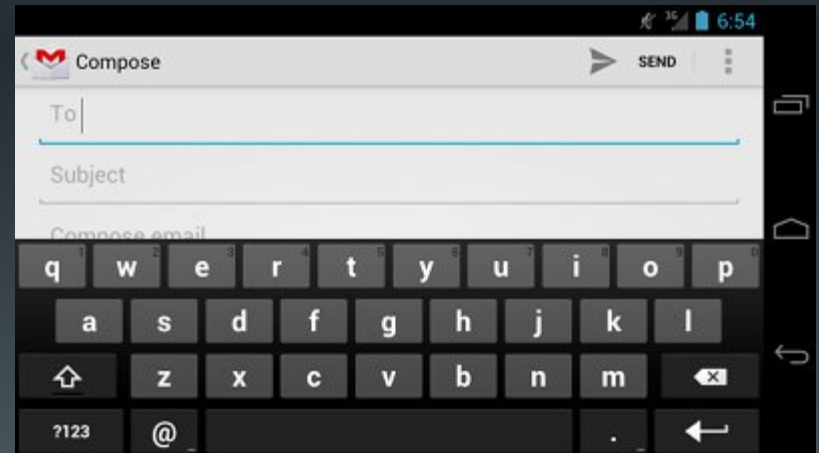
"text" : Normal text keyboard.

"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"  
  ... />
```

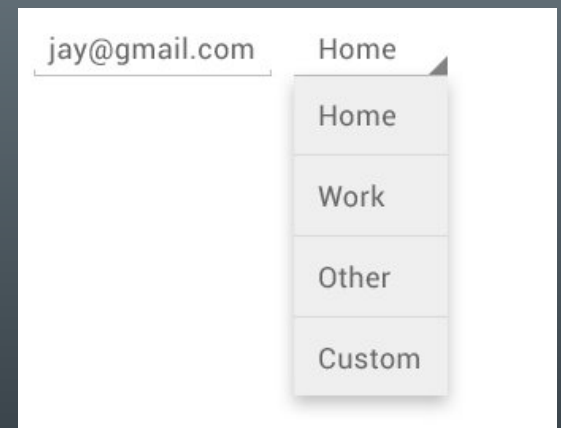
```
<Button  
  android:layout_width="wrap_content"  
  android:layout_height="wrap_content"  
  android:text="@string/button_text"  
  android:drawableLeft="@drawable/button_icon"  
  ... />
```



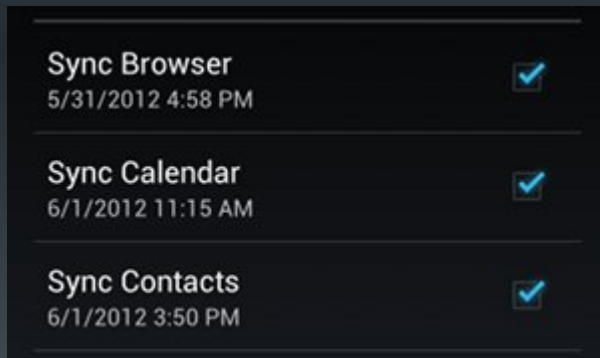
```
<ImageButton  
  android:layout_width="wrap_content"  
  android:layout_height="wrap_content"  
  android:src="@drawable/button_icon"  
  ... />
```

En vrac ...

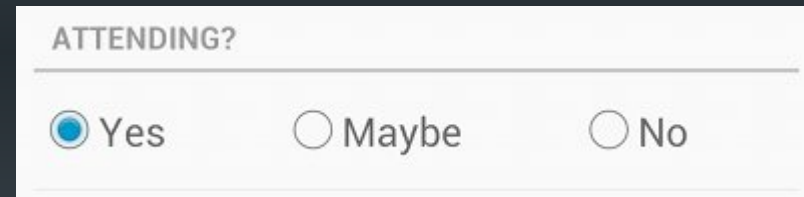
- Quelques autres widgets
 - Source developer.android.org



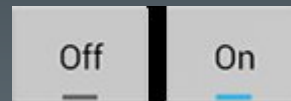
Spinner



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

Associer un « `OnClickListener` » au bouton

Surcharge de la méthode « `onClick` » de l'interface « `OnClickListener` »

Plan du cours

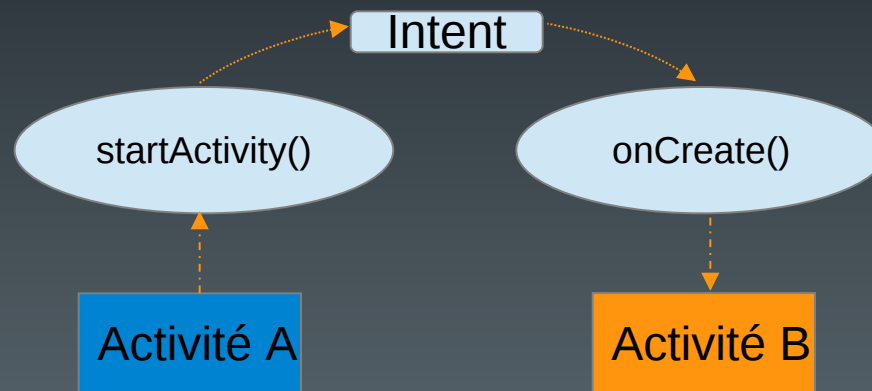
- Introduction
- Architecture d'une application Android
- Les activités
- Définir une interface graphique
- **Les intentions explicites**
- Les intentions implicites
- Les menus
- Les listes
- Les permissions
- Les content providers

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);  
    ...  
}
```

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

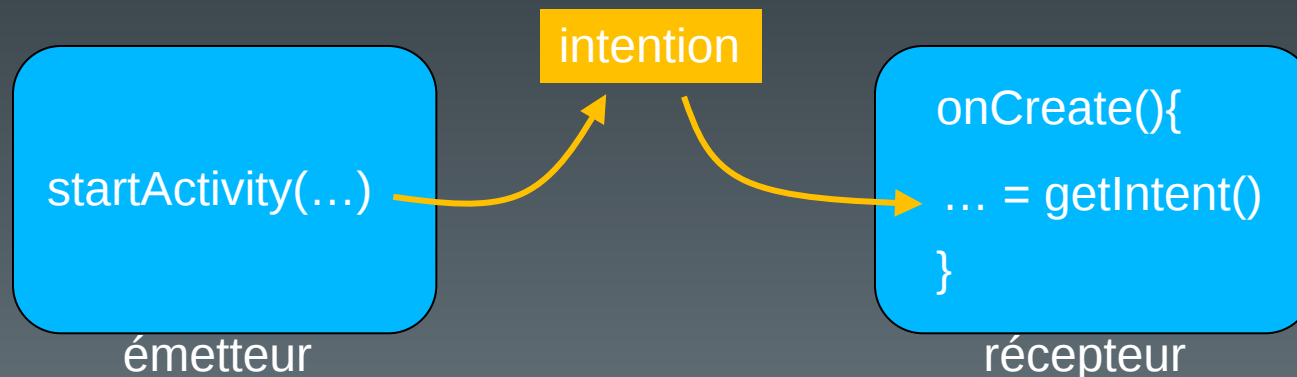
```
protected void onActivityResult (int requestCode, int resultCode, Intent data){  
    ...  
}
```

Déprécié
depuis
android 12

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)

```
...  
Intent intention1 = new Intent (...);  
...  
startActivityForResult(intention1, rCode1);
```

```
...  
Intent intention2 = new Intent (...);  
...  
startActivityForResult(intention2, rCode2)
```

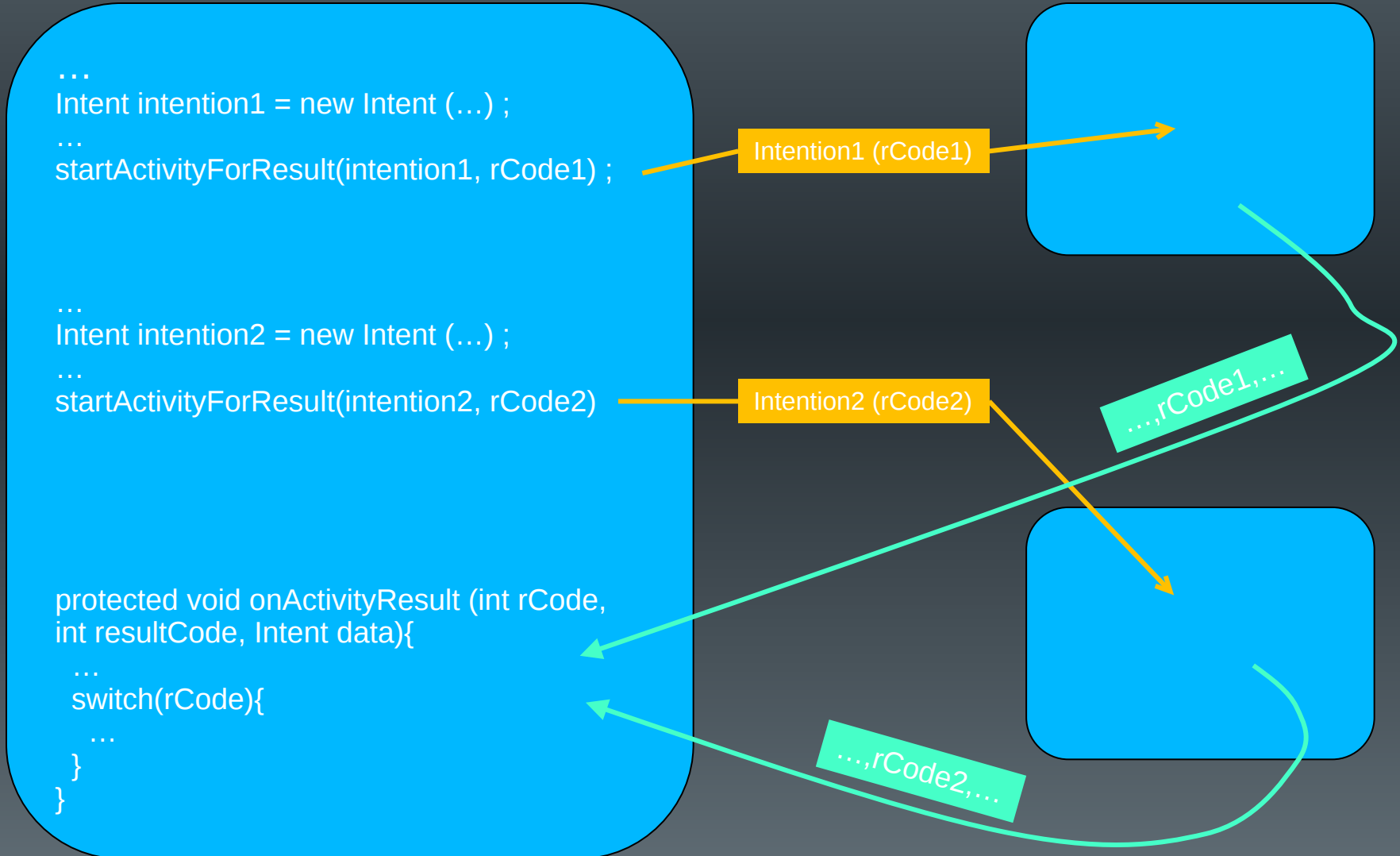
```
protected void onActivityResult (int rCode,  
int resultCode, Intent data){  
...  
switch(rCode){  
...  
}  
}
```

Intention1 (rCode1)

Intention2 (rCode2)

...,rCode1,...

...,rCode2,...



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)

```
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

(3)

(1)

(2)

(3)



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)

```
public void onCreate (Bundle savedInstanceState)  
{  
    // récupérer l'intent  
    Intent intention = getIntent () ;  
    ...  
  
    Intent resultat = new Intent(Activité1);  
    // ajout des résultats  
    ...  
    // renvoyer les résultats  
    startActivity(resultat);  
}
```

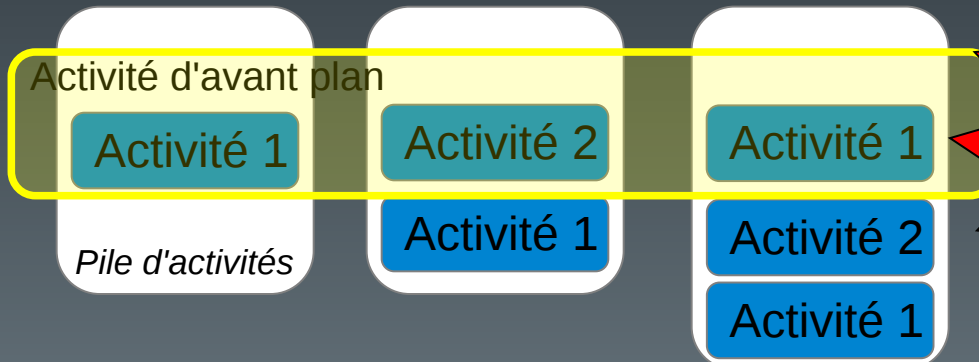
Activité2

(3)

(1)

(2)

(3)



Incorrect !!!

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()`

- Cas des intentions explicites

Ce que reçoit le lanceur

```
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(),

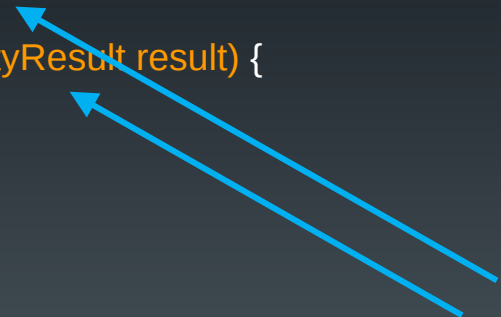
        // callback pour une intention explicite
        new ActivityResultCallback<ActivityResult>() {
            @Override
            public void onActivityResult(ActivityResult result) {

                }// onActivityResult

            }// ActivityResultCallback

        );// registerForActivityResult

    protected void onCreate(Bundle savedInstanceState) {
        ...
    }
    ...
}
```



Ce que récupère la fonction callback

Intentions avec résultats (5)

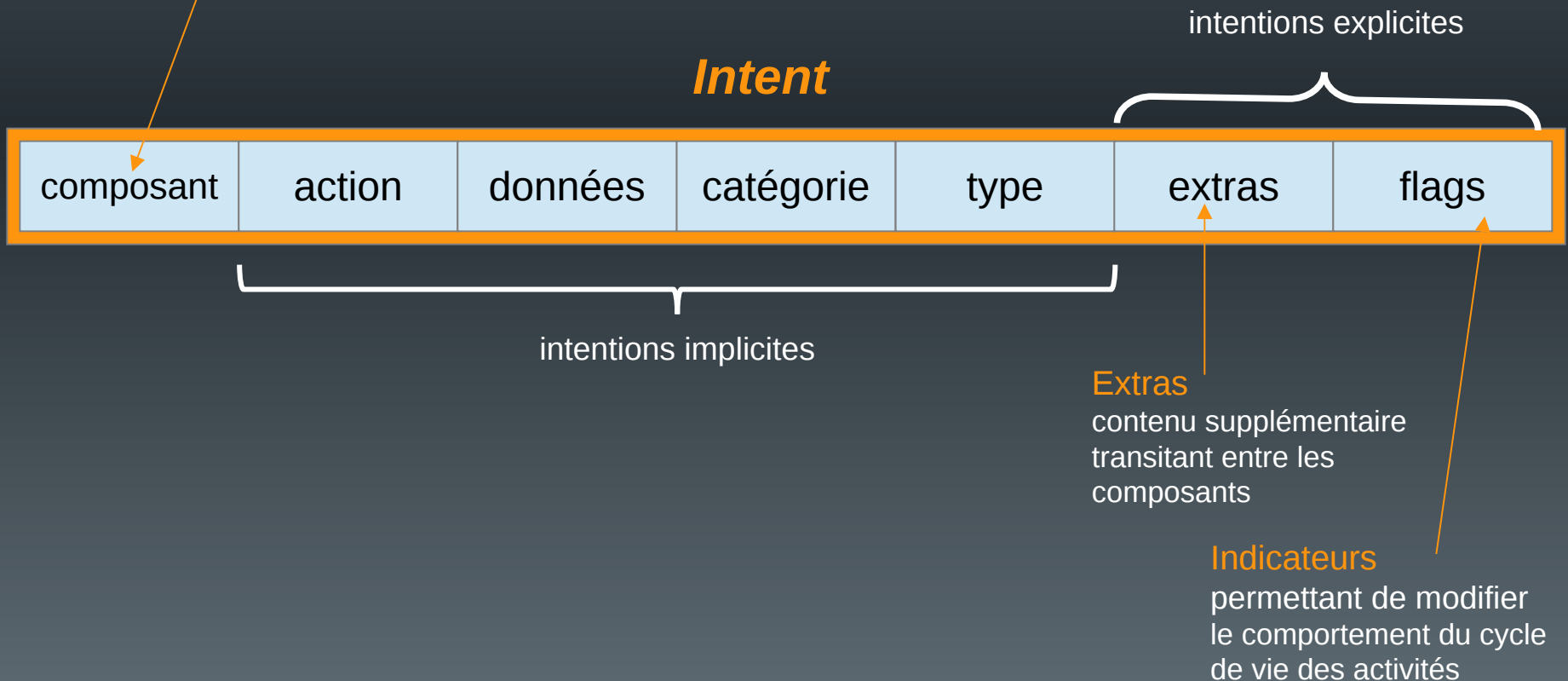
- Cas des intentions explicites
 - Récupérer les résultats

```
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



Spécifier le nom du composant

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

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

Le contexte

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

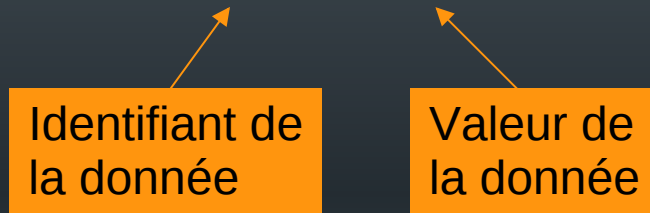
Le package

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

La classe

Les extras (1)

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



insertion

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

récupération

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

```
type get{type}Extra(String cle, type vdefault);
```

- Clé définie par l'utilisateur

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

- Quelques clés prédéfinies

- vdefault : 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.PUISS, 0) ;  
    String m = intention.getStringExtra(MainActivity.MARQUE) ;  
    ...  
}
```

Les extras (3)

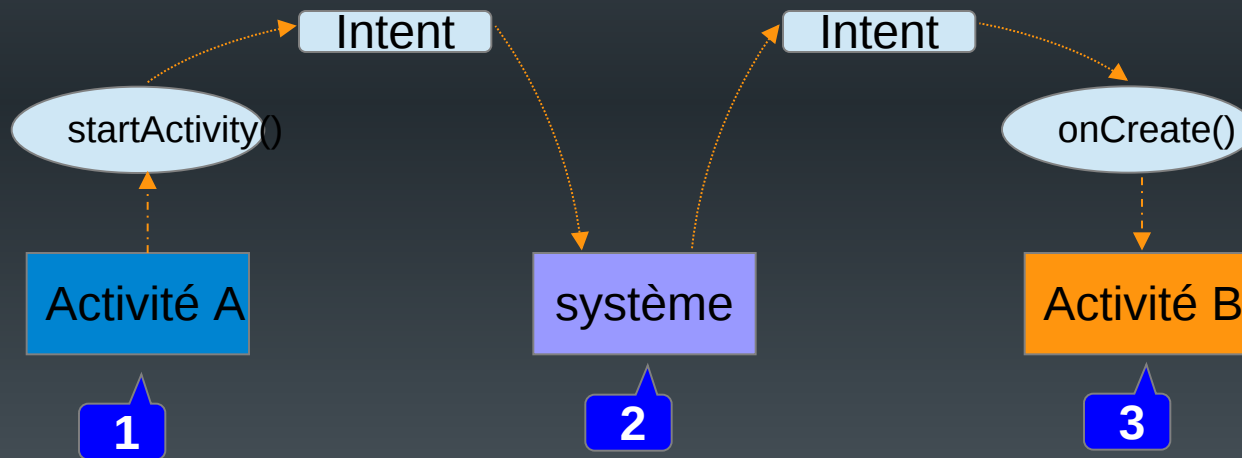
- Possibilité d'ajouter un Bundle :
 - Intent.`putExtras`(String cle, Bundle valeur) ;
 - Bundle `getBundleExtras`(String cle) ;
- Possibilité d'ajouter une intention :
 - Intent.`putExtras`(Intent valeur) ;
 - Recopie tous les extras de « valeur » dans l'intent appelant
- Possibilité d'ajouter des objets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

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 implicites

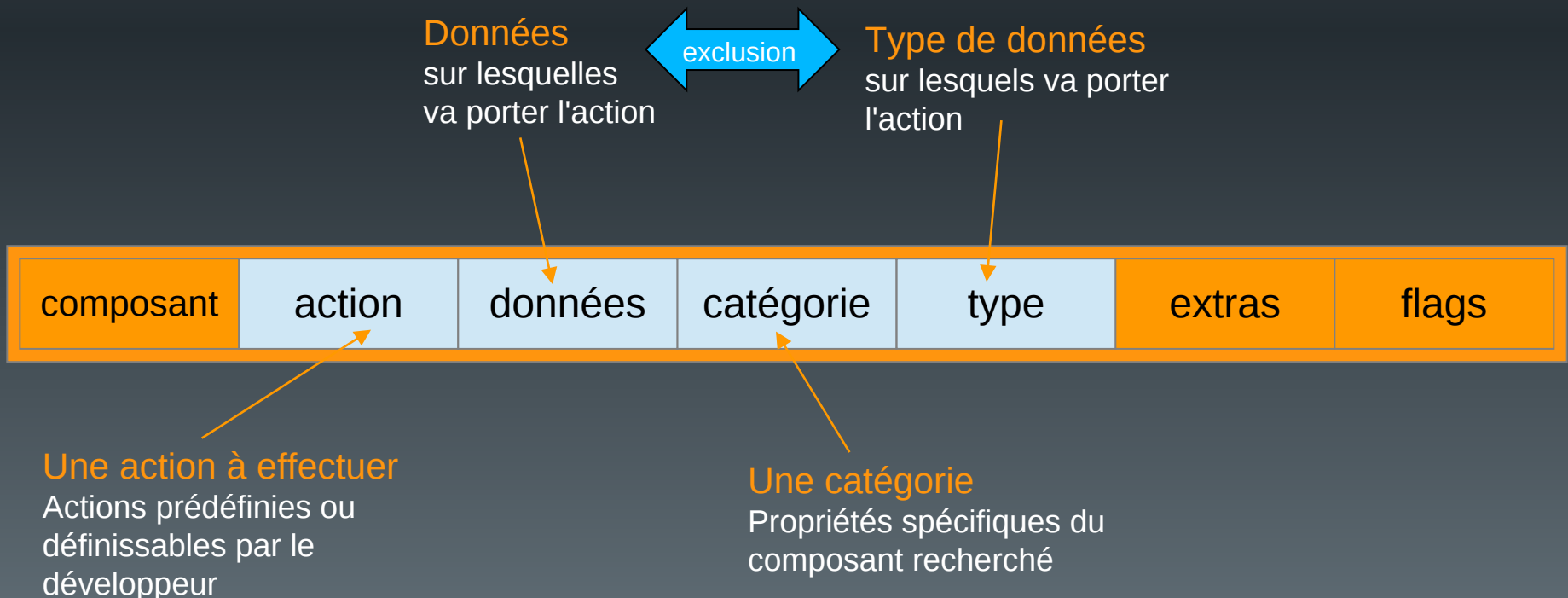
- Message à destination d'un composant inconnu
 - Le système se charge de trouver le composant adéquat et le démarre



- En cas de possibilités multiples, affichage des choix à l'utilisateur

Les Intentions implicites

- Message à destination d'un composant inconnu
 - On ne peut pas préciser son nom
 - Utilisation de méta-données permettant au système de sélectionner les applications utilisables



Action à effectuer

- Nombreuses constantes prédéfinies

- Dans la classe Intent :

- ACTION_MAIN : démarre l'activité comme point d'entrée principal
- ACTION_VIEW : affiche les données fournies à l'utilisateur
- ACTION_EDIT : permet l'édition des données fournies par l'utilisateur
- ACTION_SEND : permet l'envoi des données (mail, réseau social, ...)
- etc

- Exemples :

```
Intent intention = new Intent();  
intention.setAction(Intent.ACTION_EDIT);
```

```
Intent intention = new Intent(Intent.ACTION_EDIT, ...);
```

- Possibilité de définir ses propres actions

- Comme constante de classe

```
static final String ACTION_BIDON = "fr.univ-littoral.renaud.bidon.BIDON";
```

type Nom Valeur, incluant le nom du package

Catégorie

- Précise le type de composant qui doit prendre en charge l'intention
- Exemples :
 - CATEGORY_LAUNCHER : le composant doit être visible dans le lanceur d'applications
 - CATEGORY_CAR_MODE : le composant doit être utilisable en mode conduite de véhicule
- Remarques
 - Nombre quelconque de catégories dans une intention
 - La plupart des intentions ne nécessitent pas de catégories

```
Intent intention = new Intent();  
intention.addCategory(Intent.CATEGORY_CAR_MODE);  
intention.addCategory(Intent.CATEGORY_LAUNCHER);
```

Données

- Formatées à l'aide des URI (*Uniform Resource Identifier*)
 - Chaîne de caractères représentant un endroit ou une ressource
 - Syntaxe normalisée :

<schéma> : <information> { ? <requêtes> } { # <fragment> }

Précision quant à l'information

Nature de l'information :

http, https
content
geo
file
tel , voicemail
sms,smsto, mms, mmsto

L'information :

- Dépend du schéma
- Exemples :

tel:060000007
geo:123.456,12.3456
http://www.google.fr

Accès à une sous-partie de l'information

```
Intent intention = new Intent() ;  
Uri sms = Uri.parse("sms:060000007") ;  
intention.setData(sms) ;
```

Le type

- Normalement implicite dans les données
- Absence de données/recherche plus précise :
 - peut être précisé en utilisant les codes MIME
 - text, audio, video
 - ...
- Et des sous-types
 - text/plain, text/xml, text/html, ...
 - audio/mp3, audio/ogg, audio/wav, ...
 - Image/jpeg, image/png, ...
 - video/mpeg, video/mp4, ...

```
Intent intention = newIntent() ;  
...  
intention = intention.setType("audio/mp3") ;  
...  
String mime = intention.getType() ;
```

Élimine les données ...
(setData élimine le type)

Lancement (sans résultats)

- Similaire aux intentions explicites :

```
Intent intention = new Intent();
Intention.setAction(Intent.ACTION_VIEW);
Intention.setType("audio/*");
startActivity(intention);
```

- Gérer l'absence de composants adéquats :

```
Intent intent = new Intent() ;
...
if (intent.resolveActivity(getPackageManager()) != null) {
    startActivity(intent);
 }else{
    ...
 }
```

Utilisation du gestionnaire de paquets

```
Intent intent = new Intent() ;
...
try {
    startActivity(intent);
 } catch (ActivityNotFoundException e) {
    // aucune activité ne répond à la demande
 }
```

Utilisation des exceptions

Lancement (avec résultats) (1)

- Par `startActivityForResult` (**déprécié**)
 - Identique aux activités explicites
 - Récupération des résultats dans le champ "données" de l'intention résultats
 - On récupère un « lien » vers les données

```
Uri uri = data.getData();
```

Lancement (avec résultats) (2)

- Par classe de contrat (à partir de Android 12)

Type de ce qu'on l'on va envoyer Dépend de la classe de contrat

```
public class MainActivity extends AppCompatActivity {
    // attributs de la classe
    private ActivityResultLauncher<TYPE> mGetResult = registerForActivityResult(
        // classe de contrat pour une intention implicite
        new ActivityResultContracts.XXX(),
        // callback pour une intention explicite
        new ActivityResultCallback<TYPE_RESULTAT>() {
            @Override
            public void onActivityResult(TYPE_RESULTAT result) {
                // onActivityResult
            }
        }
    ); // ActivityResultCallback
}; // registerForActivityResult
}
```

Type de ce qu'on l'on va récupérer

Lancement (avec résultats) (3)

- Nombreuses classes de contrat
 - `ActivityResultContracts.StartActivityForResult`
 - Cf activités explicites
 - `ActivityResultContracts.CaptureVideo`
 - Retourne l'uri d'une vidéo faite par l'utilisateur
 - `ActivityResultContracts.GetContent`
 - Retourne une uri d'une donnée choisie par l'utilisateur
 - `ActivityResultContracts.PickContact`
 - Retourne un contact choisi par l'utilisateur
 - ...

Lancement (avec résultats) (4)

- Un exemple :
 - Choisir un fichier stocké sur le périphérique (image, son, vidéo, etc...)
- Quelle classe de contrat ?

– `ActivityResultContracts.GetContent`

`myLauncher.launch("audio/*")`

Qu'envoie-t-on ?

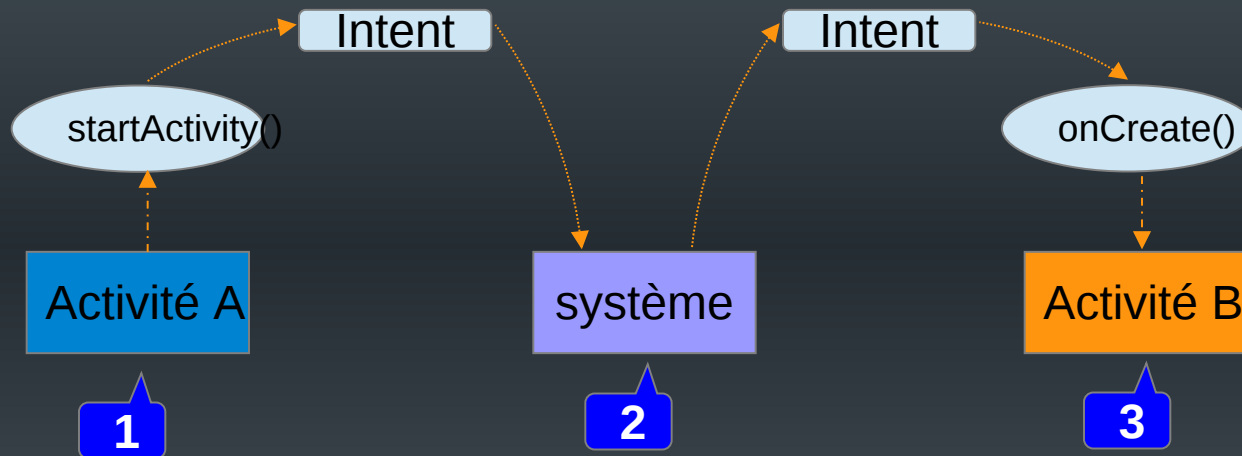
le type mime du fichier

```
ActivityResultLauncher< String > myLauncher = registerForActivityResult(  
    new ActivityResultContracts.GetContent(),  
    new ActivityResultCallback< Uri >() {  
        @Override  
        public void onActivityResult( Uri resultat ) {  
            ...  
        }  
    });
```

Que récupère t-on ?

Recherche d'un composant

- Utilisation d'intentions implicites :
 - Le système recherche le(s) composant(s) possible(s)

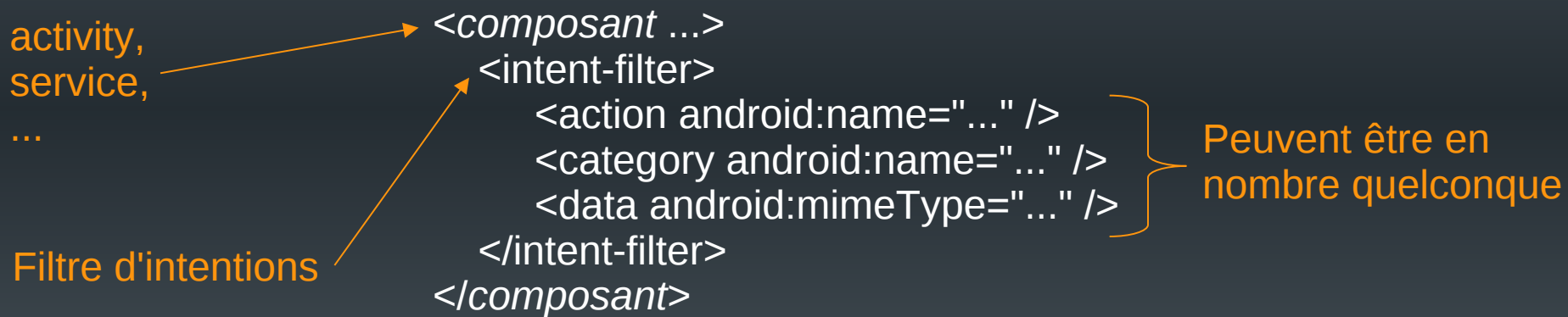


- Le système doit connaître les types d'intentions que peut recevoir chaque composant
 - Nécessité de préciser pour chaque composant les intentions réceptionnables

➔ *Intent filters*

Les intent filters

- Présents dans le fichier *manifest.xml* pour chaque composant d'une application
- Syntaxe :



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

Composant principal
d'une application

Doit apparaître dans la liste
des applications pouvant être lancées

Remarques

- Intentions et services :
 - Démarrage d'un service :
 - N'utiliser que des intentions explicites
 - Intentions implicites : l'utilisateur ne sait pas forcément quel service est lancé et il n'a pas de retour visuel
 - Pas d'*intent filter* pour un service

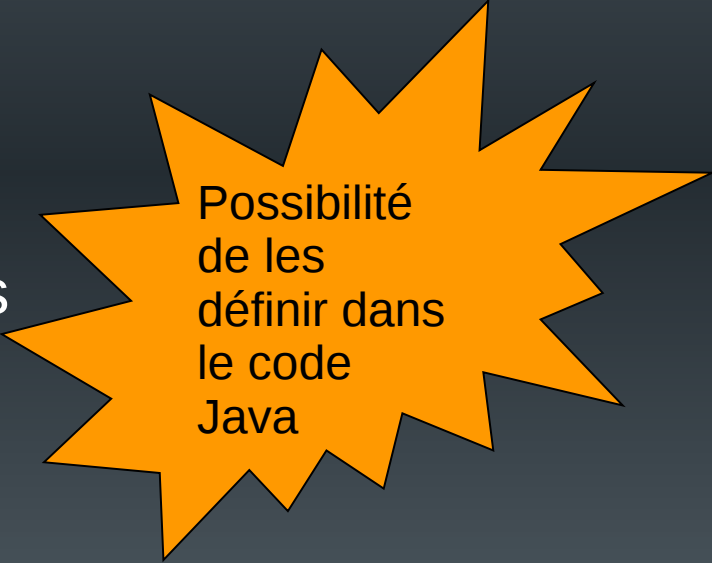


Plan du cours

- 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 menus

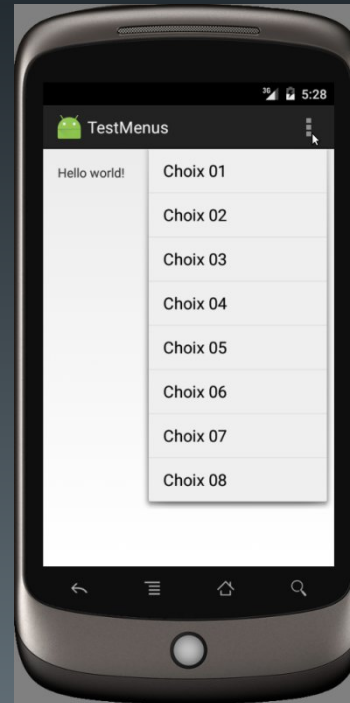
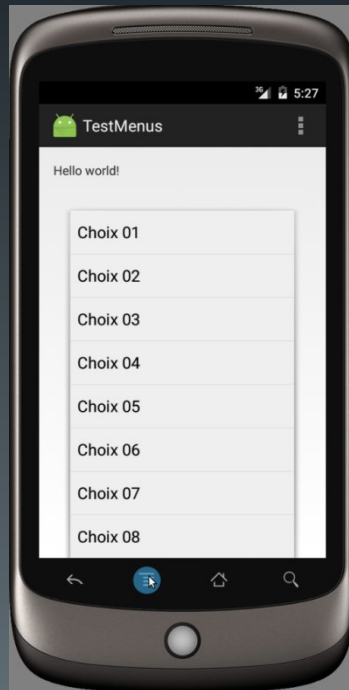
- Disponibles pour chaque activité
- Trois types de menus :
 - Le menu d'options
 - Les menus contextuels
 - Les menus pop-ups
- Considérés comme des ressources
 - À définir en XML (dossier res/menus)
 - Facilite la maintenance des menus
 - séparation de la définition du code de gestion
 - Possibilité de créer des menus multi-plateforme
 - Facilite la visualisation de la structure des menus



Possibilité
de les
définir dans
le code
Java

Le menu d'options (1)

- Accessibles via :
 - le bouton de « menu »
 - N'est plus forcément disponible sur les périphériques récents
 - l'icône « menu » de la barre d'application



Le menu d'options (2)

- Définition XML

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"  
      xmlns:tools="http://schemas.android.com/tools"  
      tools:context=".MainActivity">
```

```
<item android:id="@+id/choix01"  
      android:title="@string/texte_choix01" />
```

...

```
<item android:id="@+id/choixNN"  
      android:title="@string/texte_choixNN" />
```

```
</menu>
```

Identifiant de
chaque entrée

Libellé du choix



Le menu d'options (3)

- Attributs des items

```
<item android:id="@+id/choix01"  
      android:title="@string/texte_choix01"  
      android:icon="..."  
      android:onClick="..."  
      android:showAsAction=" " />
```

Icône associée au choix

Callback en cas de choix

Précise quand et comment l'item peut apparaître dans la barre d'application :

- "ifRoom" : s'il y a de la place
- "never" : jamais
- "always" : toujours (à éviter sauf item important pour des raisons de place)
- ...

Le menu d'options (4)

- Chargement du menu
 - Appel de la méthode `onCreateOptionsMenu()` au lancement de l'application

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.monMenu, menu);

    return true;
}
```

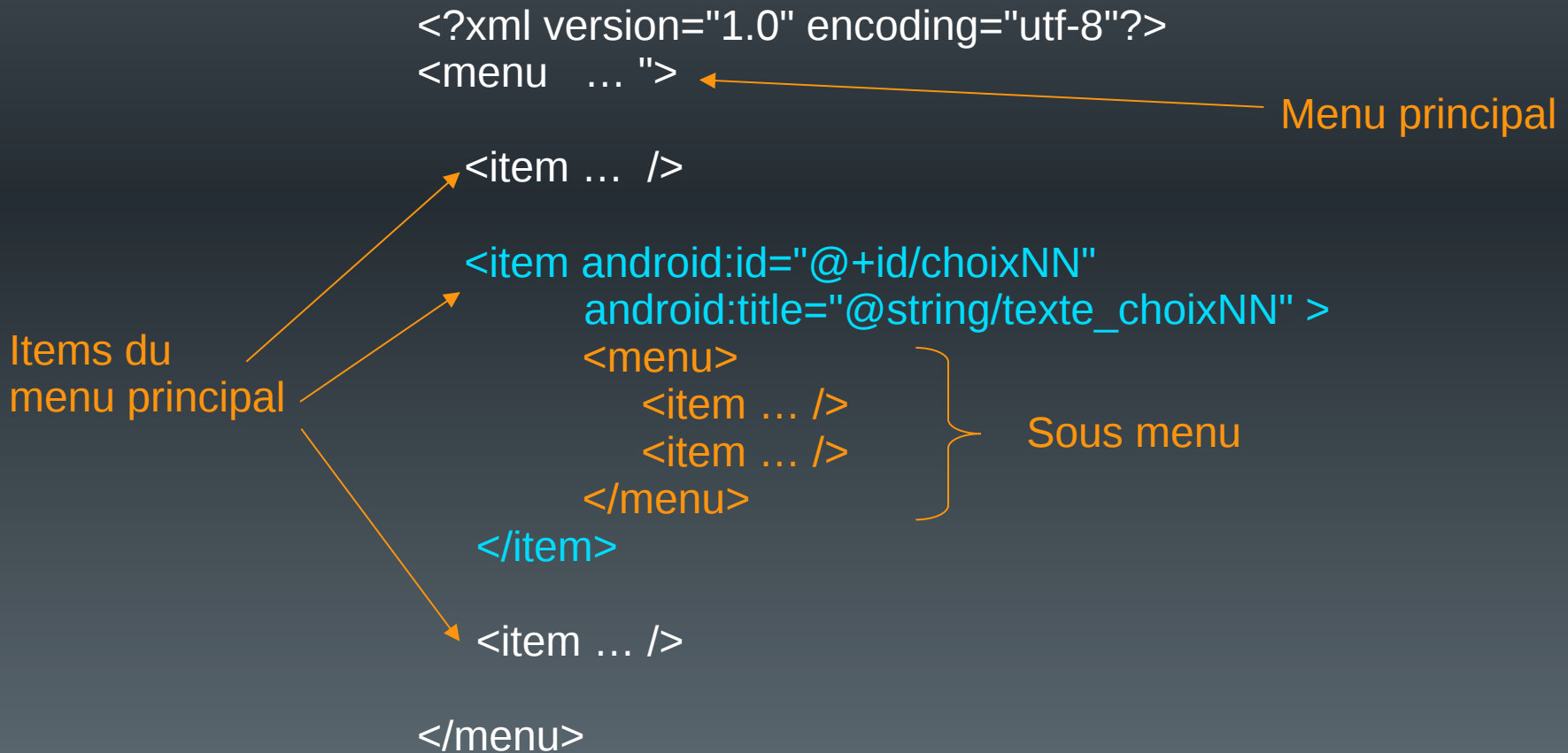
Menu de l'application
devant être initialisé

Objet permettant de gérer
l'association XML/menu

Association XML/menu

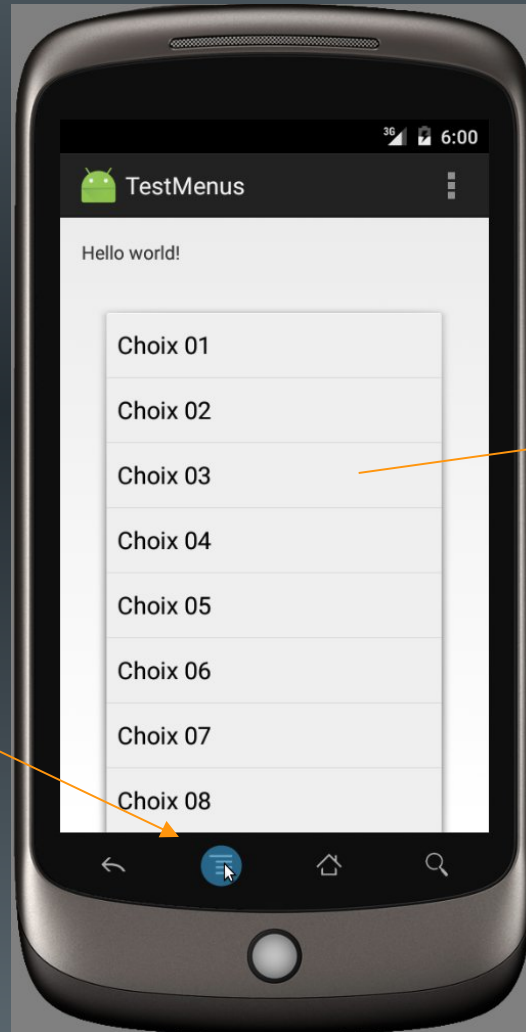
Le menu d'options (6)

- Extensions aux sous-menus
 - tout item peut contenir un sous-menu

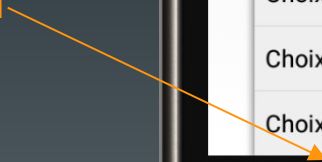


Le menu d'options (7)

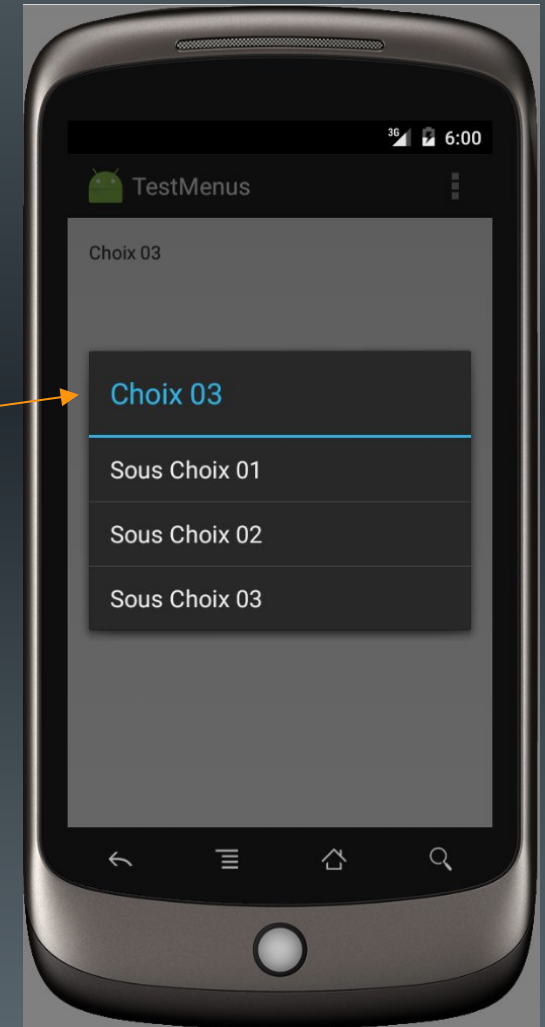
- Exemples



Sélection du menu principal

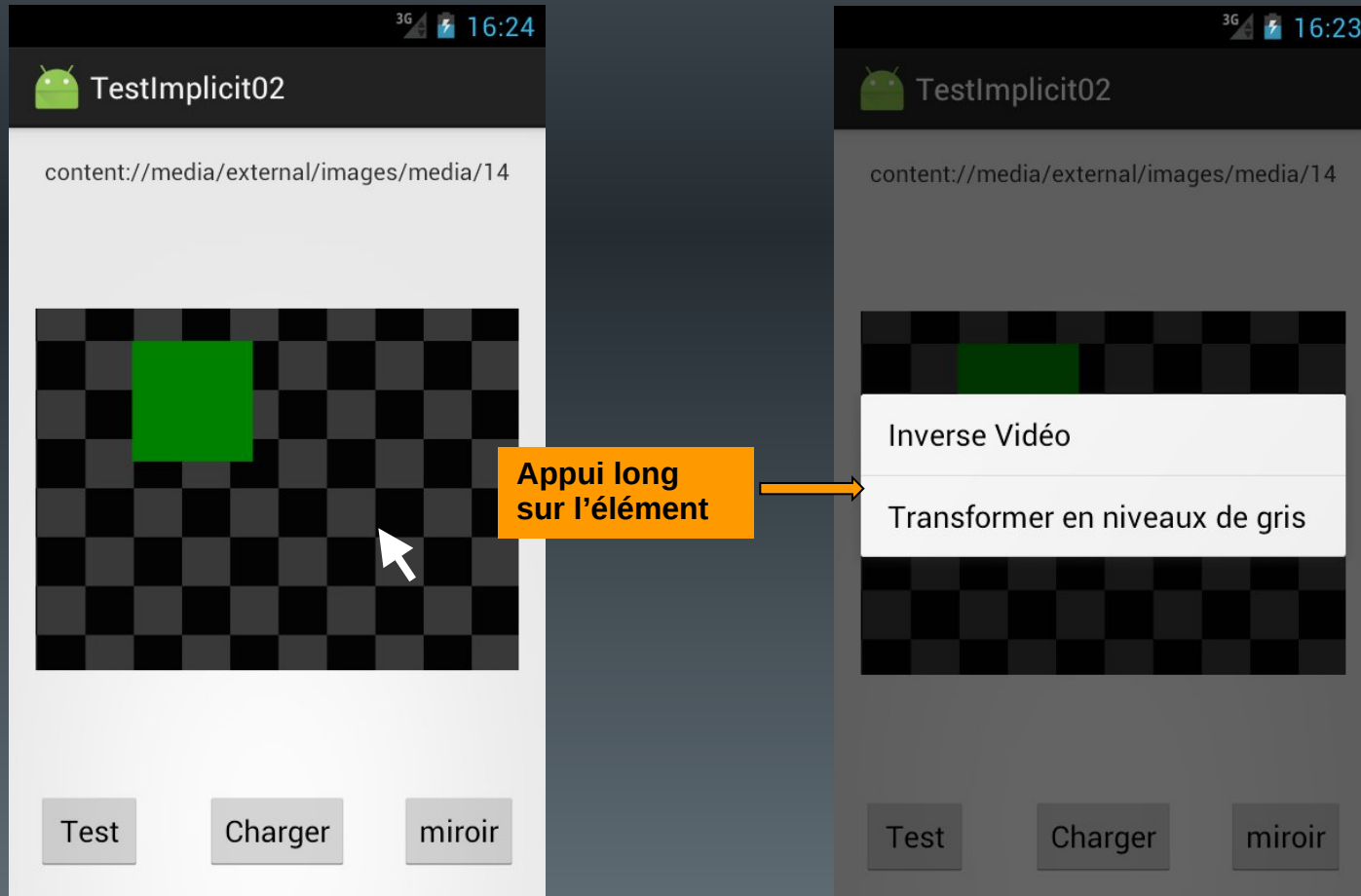


Sélection d'un sous-menu



Les menus contextuels (1)

- Associables à tout élément de l'interface graphique



Les menus contextuels (2)

- Utilisent un fichier XML décrivant un menu

1. Enregistrer l'élément pour lequel un menu contextuel doit être créé

```
registerForContextMenu(View view)
```

2. Surcharger la méthode onCreateContextMenu()

```
@Override  
public void onCreateContextMenu(ContextMenu menu, View v,  
                               ContextMenuInfo menuInfo) {  
  
    super.onCreateContextMenu(menu, v, menuInfo);  
  
    MenuInflater inflater = getMenuInflater();  
    inflater.inflate(R.menu.context_menu, menu);  
}
```

*Nécessaire si
plusieurs View
utilisent un menu
contextuel ...*

*Informations
complémentaires*

*Ex. item d'une liste
ayant activé le menu*

Les menus contextuels (3)

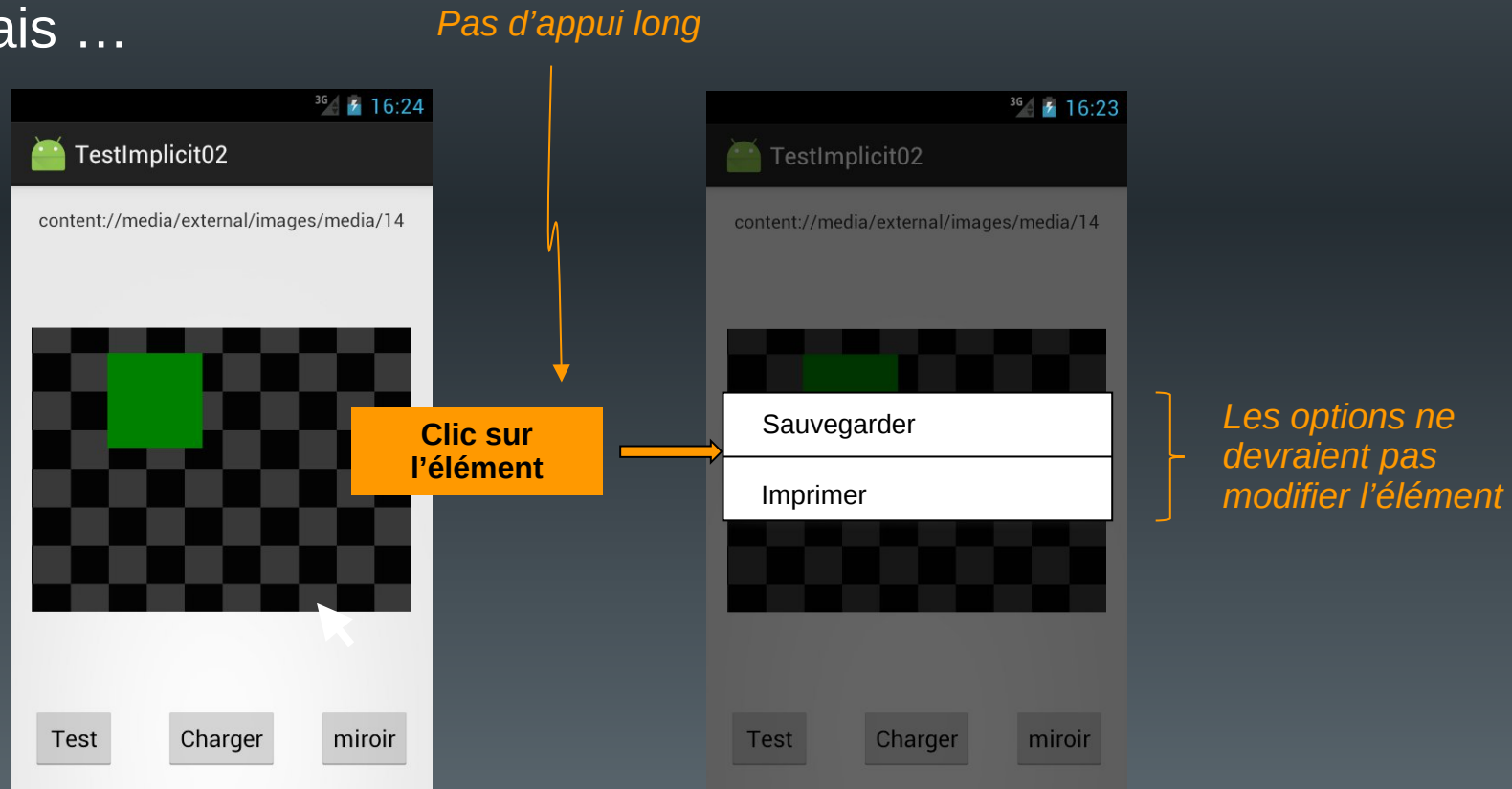
3. Surcharger la méthode onOptionsItemSelected()

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {

    switch (item.getItemId()) {
        case R.id.xxx:
            ...
            return true;
        case R.id.yyy:
            ...
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Les menus popup (1)

- Associables à tout élément de l'interface graphique
- Très similaires aux menus contextuels
 - Mais ...



Gestion « manuelle » de l'événement ...

Les menus popup (2)

(1) Créer le menu (XML)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/popup01"
        android:title="@string/menu_popup01" />
  <item android:id="@+id/popup02"
        android:title="@string/menu_popup02" />
</menu>
```

(2) Associer une méthode de gestion du clic au widget

```
< Widget
  android:id="@+id/name"
  ...
```

(3) Créer la méthode d'ouverture du popup

```
public void showPopupX(View v) {
  PopupMenu popup = new PopupMenu(this, v);
  MenuInflater inflater = popup.getMenuInflater();
  inflater.inflate(R.menu.menu_popup, popup.getMenu());
  popup.inflate(R.menu.menu_popup);
```

```
/>
```

```
  android:onClick="showPopupX"
```

Créer le popup

```
  popup.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
      switch (item.getItemId()) {
        case R.id.popup01:
          ...
      }
    }
  });
  popup.show();
}
```

Implémenter l'interface OnMenuItemClickListener

Associer un gestionnaire de clic

Afficher le popup

TP 05



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 listes

- Éléments très utilisés sous android
 - Facilitent l'affichage et la manipulation d'ensembles de données importants sur des écrans de taille réduite
 - Possibilité de faire défiler/sélectionner les items
 - Paramétrage possible de la présentation des items via un layout
 - Remplissage « automatique » à partir de données extraites de tableaux ou de bases de données
 - Nécessitent l'utilisation d'un **adaptateur**

Examples

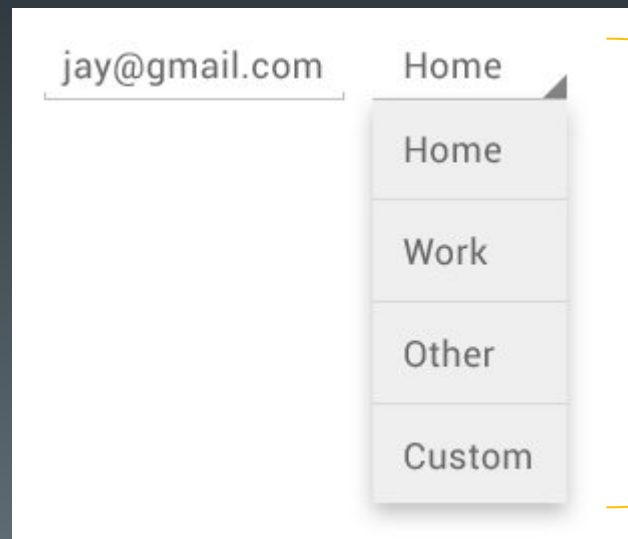


ListView

GridView

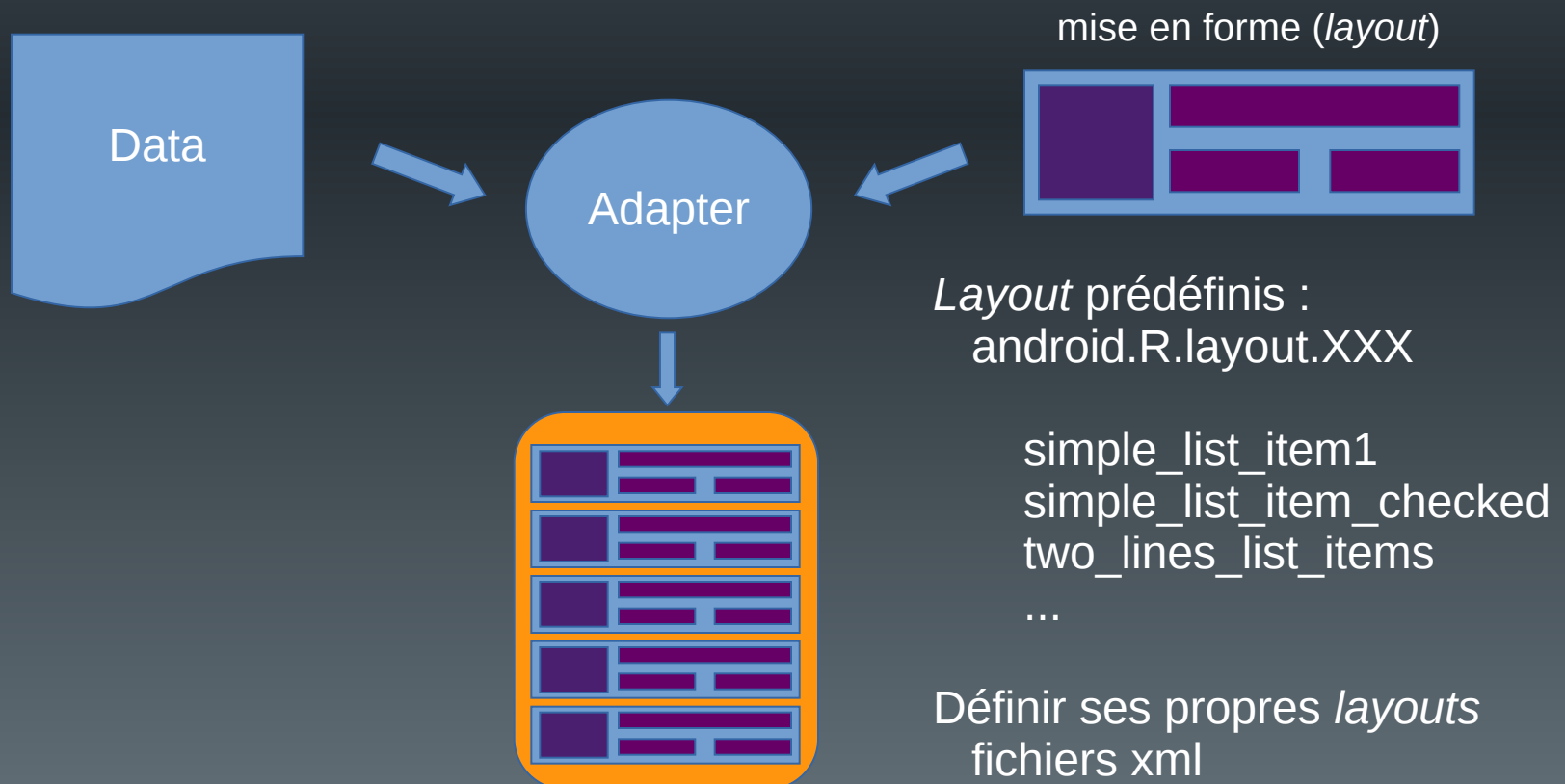


Spinner



Les adaptateurs (1)

- Rôle
 - Adapter les données au format d'affichage
 - Nécessité de préciser le layout utilisé pour les items



Les adaptateurs (2)

- Différentes classes prédéfinies
 - **ArrayAdapter** : utilise un tableau d'objets quelconques ou une liste java
 - **SimpleCursorAdapter** : utilise des ensembles de données issus de requêtes sur des bases de données
 - **SimpleAdapter** : utilise des maps {(clé, valeur),...}

Le contexte de l'activité (this, getContext())

```
ArrayAdapter(Context c, int ressource) ;  
ArrayAdapter(Context c, int ressource, T[] objets) ;
```

L'identifiant du fichier ressource
décrivant le layout à utiliser
(R.layout....)

Les objets à insérer dans
une liste via l'adaptateur

Les adaptateurs (3)

- Quelques méthodes utiles (ArrayAdapter)
 - `int getCount()` : nb d'objets gérés par l'adaptateur
 - `add(T objet)` : ajoute l'objet à la fin de la liste
 - `insert(T objet, int position)` : insertion de l'objet à une position donnée
 - `remove(T objet)` : suppression de l'objet de la liste
 - `clear()` : suppression de tous les objets de la liste
 - `getItem(int position)`
- Remarque :

```
ArrayAdapter(Context c, int ressource, T[] objets) ;
```

 Les objets insérés deviennent "*immutable*"
impossible d'en ajouter ou d'en supprimer ...

Les ListView (1)

- Composant graphique qui gère l'affichage d'une liste
 - Nécessite un adaptateur

- Configuration XML

```
<ListView
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:id="@+id/maListe"
```

```
    android:choiceMode="none" />
```

none : éléments non sélectionnables

singleChoice : 1 seul choix possible

multipleChoice : choix multiples

- Déclaration Java



L'adaptateur doit utiliser
le layout correspondant ...

```
ListView lv = (ListView)findViewById(R.id.maListe);
```

- Remplissage :

- Statique : depuis un tableau de données xml
- Dynamique : via un adaptateur

Les ListView (2)

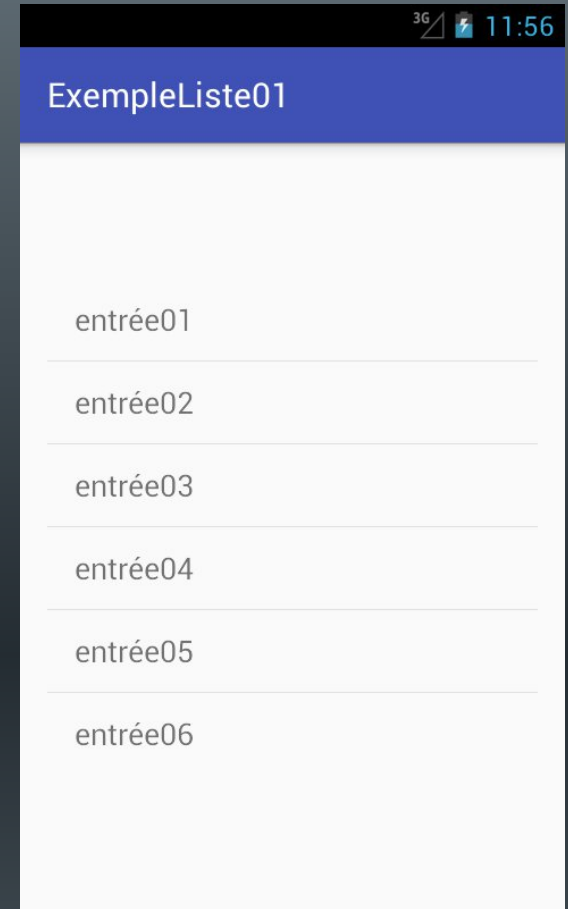
- Remplissage statique (xml)

res/values/strings.xml

```
<resources>
...
  <string-array name="maListe">
    <item>entrée01</item>
    <item>entrée02</item>
    ....
    <item>entrée06</item>
  </string-array>
...
</resources>
```

res/layout/content.xml

```
<ConstraintLayout ... >
...
  <ListView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/maListe"
    android:entries="@array/maListe" />
...
</ConstraintLayout>
```



Les ListView (3)

- Remplissage dynamique (java)

res/layout/content.xml

```
...  
<ListView  
  ...  
  android:id="@+id/maListe" />  
...
```

```
ListView lv = (ListView)findViewById(R.id.maListe);
```

```
String entrees[]={"valeur01", "valeur02", "valeur03",  
                 "valeur04", "valeur05", "valeur06"};
```

```
ArrayAdapter<String> adapter =  
    new ArrayAdapter<String>(this,  
                             android.R.layout.simple_list_item_1,  
                             entrees);
```

```
lv.setAdapter(adapter);
```

