Présentation Android

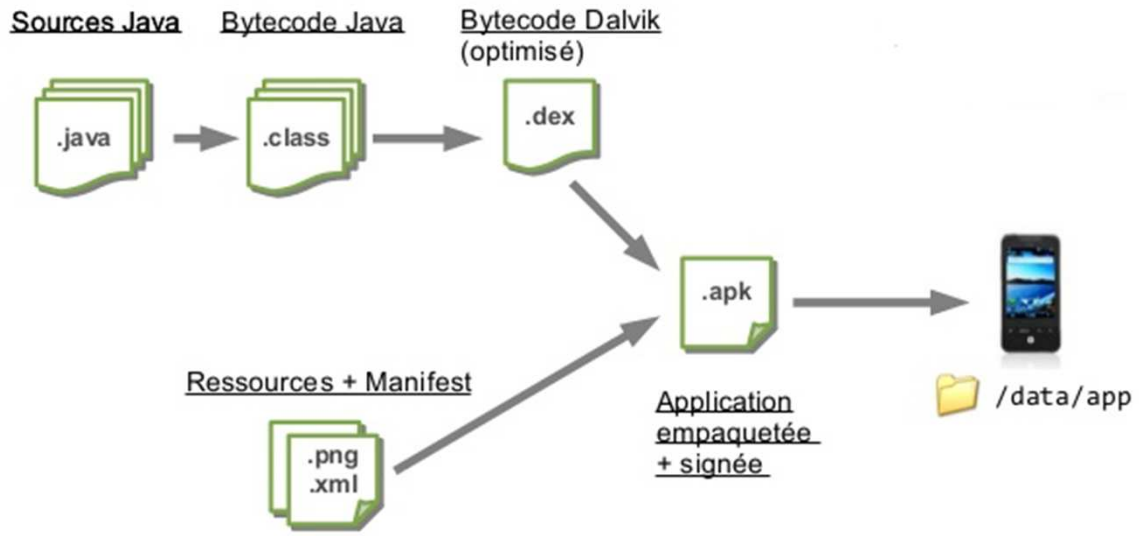
Partie 2

Les concepts de base

Concepts de base

⇨ l'application ?

- Une application Android se développe en Java avec des ressources dans des fichiers Xml.
- Une fois votre application compilée, elle tient dans un seul fichier d'extension « apk ».
- Il contient aussi bien le code java que les fichiers de ressources, les images, …



Concepts de base

⇨ les composants fondamentaux

- Les principaux sont au nombre de 4 :
  - Les activités

    Classe : Activity
  - Les services

    Classe : Service
  - Les fournisseurs de contenu

    Classe : ContentProvider
  - Les récepteurs d'informations

    Classe : BroadcastReceiver

On pourrait rajouter les intentions (Intent) qui permettent à ces composants de communiquer

la classe « Activity »

- Une activité gère l'affichage et l'interaction avec l'utilisateur.

  Elle aura donc un *Layout* qui lui-même contiendra les composants graphiques appelés *View*

- Une application peut avoir plusieurs activités qui sont indépendantes les unes des autres.
- Une activité principale permet de lancer l'application (elle est de type *launcher*).

  (Il peut y avoir plusieurs activités permettant de lancer la même application de façons différentes).

la classe « Service »

- Un service est un composant qui s'exécute en « tâche de fond ».
- Contrairement à une activité, il n'a pas d'interface graphique.
- Exemple :
  - Jouer de la musique
  - Rechercher de données sur un réseau

la classe « ContentProvider »

- C'est lui qui gère les données partageables.
- C'est le seul moyen d'accéder aux données des autres applications.
- On peut donc créer un *ContentProvider* si l'on veut partager certaines données de notre application.
- Pour récupérer un *ContentProvider* d'une autre application, il faut récupérer le *ContentResolver* du système et lui envoyer une requête avec l'URI des données.

la classe « BroadcastReceiver »

- C'est un récepteur qui est à l'écoute de certaines informations.
- C'est à lui d'indiquer les informations qui l'intéressent.
- L'application n'a pas besoin d'être lancée, Android s'en chargera si des infos arrivent.
- Un récepteur n'a pas d'interface graphique mais peut créer des notifications.
- Ces informations sont en fait des *Intents*

la classe « Intent »

- Un *Intent* est une intention à faire quelque chose.
- Il contient des informations destinées à un autre composant Android.
- C'est un message asynchrone.
- Deux types d'Intent sont à distinguer :
  - les Intents implicites

    Il ne contient pas le nom du composant qui doit le gérer. Le système Android va se débrouiller.
  - les Intents explicite

    Il contient la classe du composant qui doit le gérer.

Toute application doit contenir un fichier

AndroidManifest.xml

- Il déclare les différents composants de l'application
  - activity, service, contentProvider, …
  - les déclarations des broadcastReceiver ne sont pas obligatoires
- Il permet aussi de préciser:
  - Les permissions nécessaires (accès internet, accès aux données partagées,…)
  - Le numéro minimum de la version API
  - Les matériels utilisés (caméra, …)
  - Les bibliothèques supplémentaires (Google Maps Library, …)

Les activités

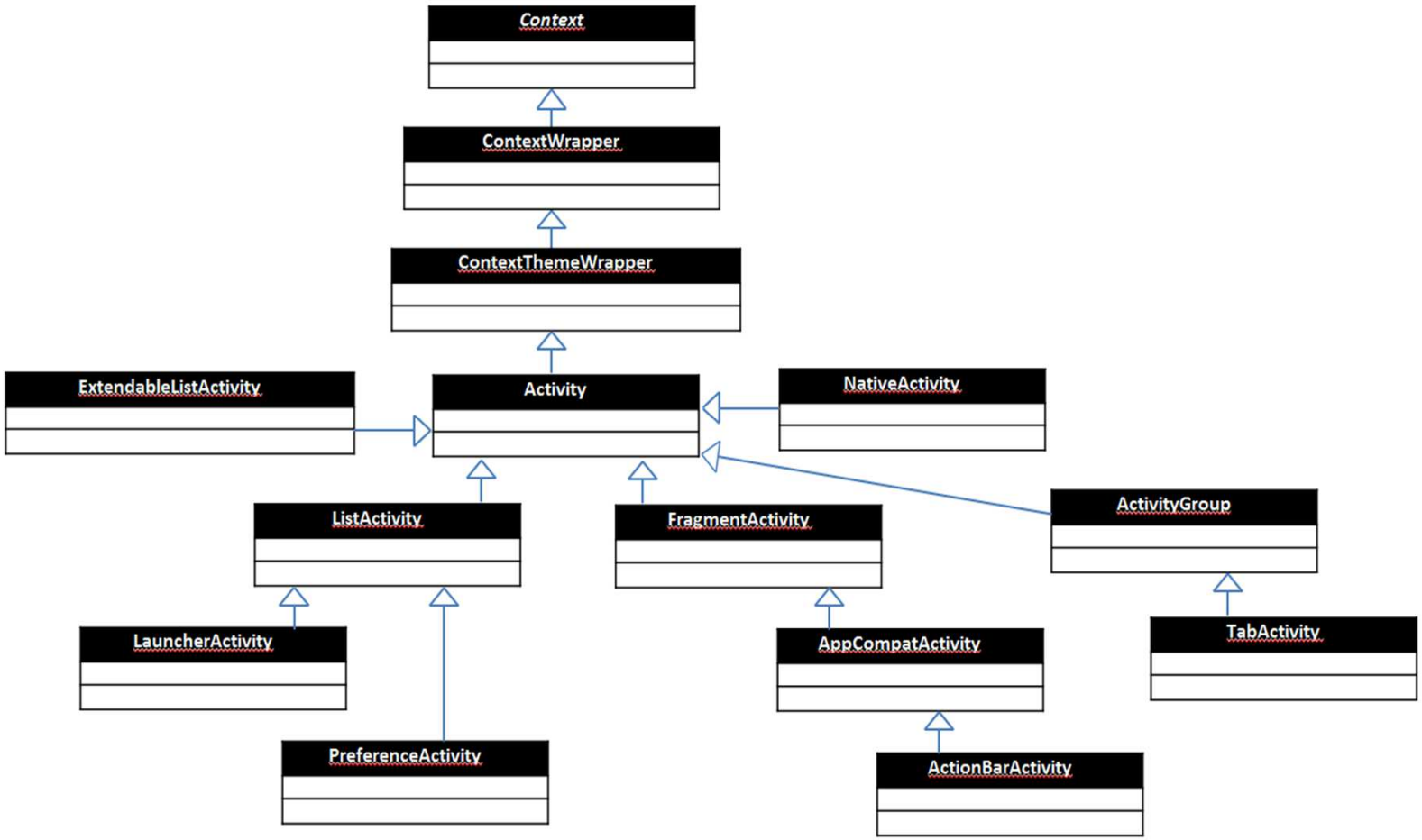
⇨ la classe Activity (1)

- La classe Activity est la classe générale.
- Il existe des classes dérivées :
  - AppCompatActivity
  - ListActivity
  - TabActivity
- Une activité à un cycle de vie (voir plus loin) et change d'état au cours du temps.
- A chaque changement d'état, une des méthodes *onXXX()* sera appelée.
- Exemple :
  - A la création de l'activité, la méthode *onCreate()* est appelée ce qui permettra d'initialiser notre application

Les activités

⇨ la classe Activity (2)

- Diagramme de classe simplifié :



Les activités

⇨ la classe Activity (3)

- Nos activités seront des classes qui devront étendre la classe Activity ou une classe dérivée (ex. AppCompatActivity).
- Import : android.app.Activity
- Plusieurs méthodes pourront être redéfinies.

```
public class MainActivity extends Activity
 {
        @Override
        protected void onCreate() { …}
        @Override
        protected void onPause() { …}
        @Override
        protected void onStop() { …}
        @Override
        protected void onDestroy() { …}
 }
```

Les activités

⇨ Déclaration (1)

- L'activité doit être déclarée dans le manifest sinon rien ne sera exécuté.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="fr.afpa.cdi.exandroid" …>

  <application ...>
    <activity android:name=".MainActivity">
      …
    </activity>

    <activity android:name=". AboutActivity">
      …
    </activity>
    ...
  </application>

</manifest>
```

Les activités

⇨ déclaration (2)

- Certaines activités sont marquées pour être démarrables de l'extérieur :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="fr.afpa.cdi.exandroid" …>

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

</manifest>
```

Remarque :

- Il peut il avoir plusieurs activités qui permettent de lancer l'application

- Exemple complet :

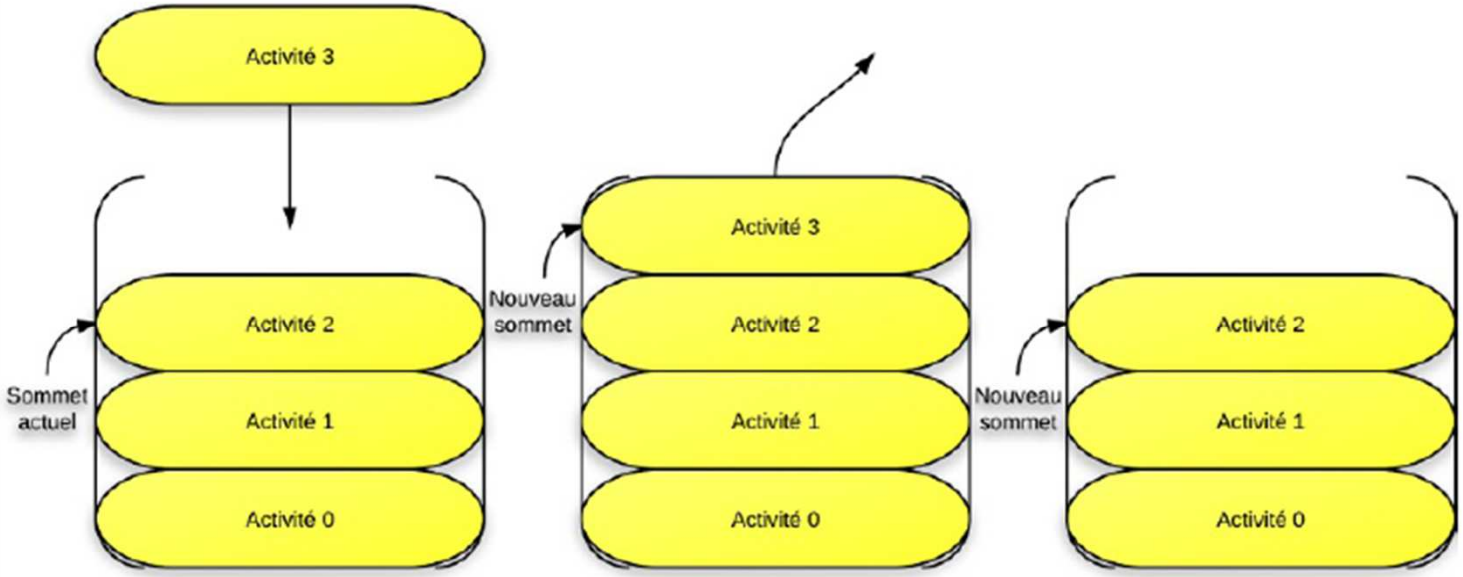
```
package fr.afpa.cdi.exandroid;
import android.app.Activity;
...
public class ExAndroidActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Nom du fichier XML de ressource décrivant l'interface graphique (voir Interface graphique plus loin)

Les activités

⇨ fonctionnement (1)

- Concrètement, Une application Android est composée d'activités.
- Une activité = 1 page, avec des vues (TextView, EditText, Button...)
- Android gère une sorte de pile d'activités, seule celle du premier plan est active

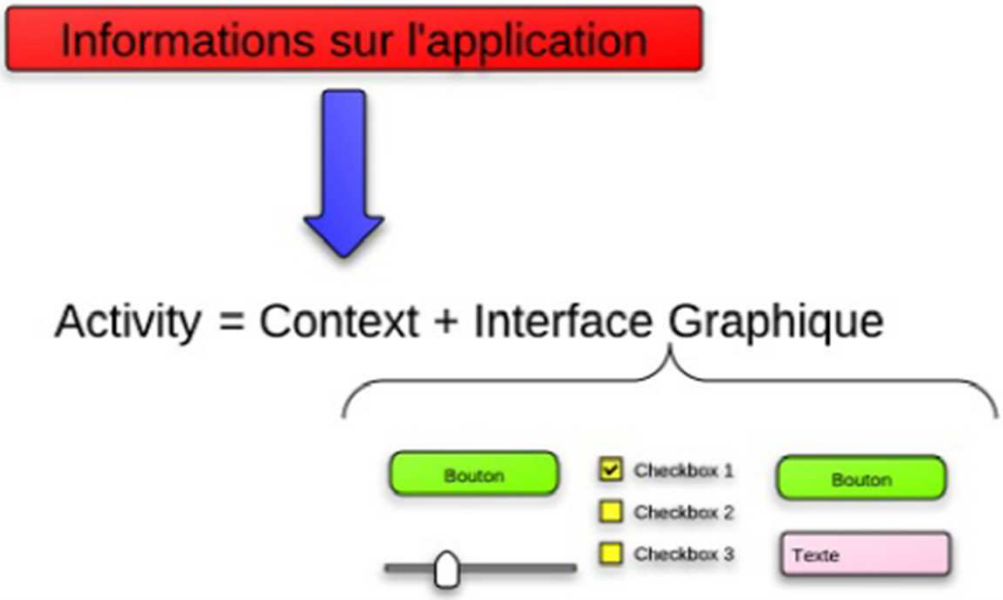


Fonctionnement de la pile d'activités

Les activités

⇨ fonctionnement (2)

- En fait, une activité contient des informations sur l'état de l'application appelé le ***context***



- Ce *context* sera utile dans beaucoup de cas.
- La classe Activity dérive de la classe Context

Les activités

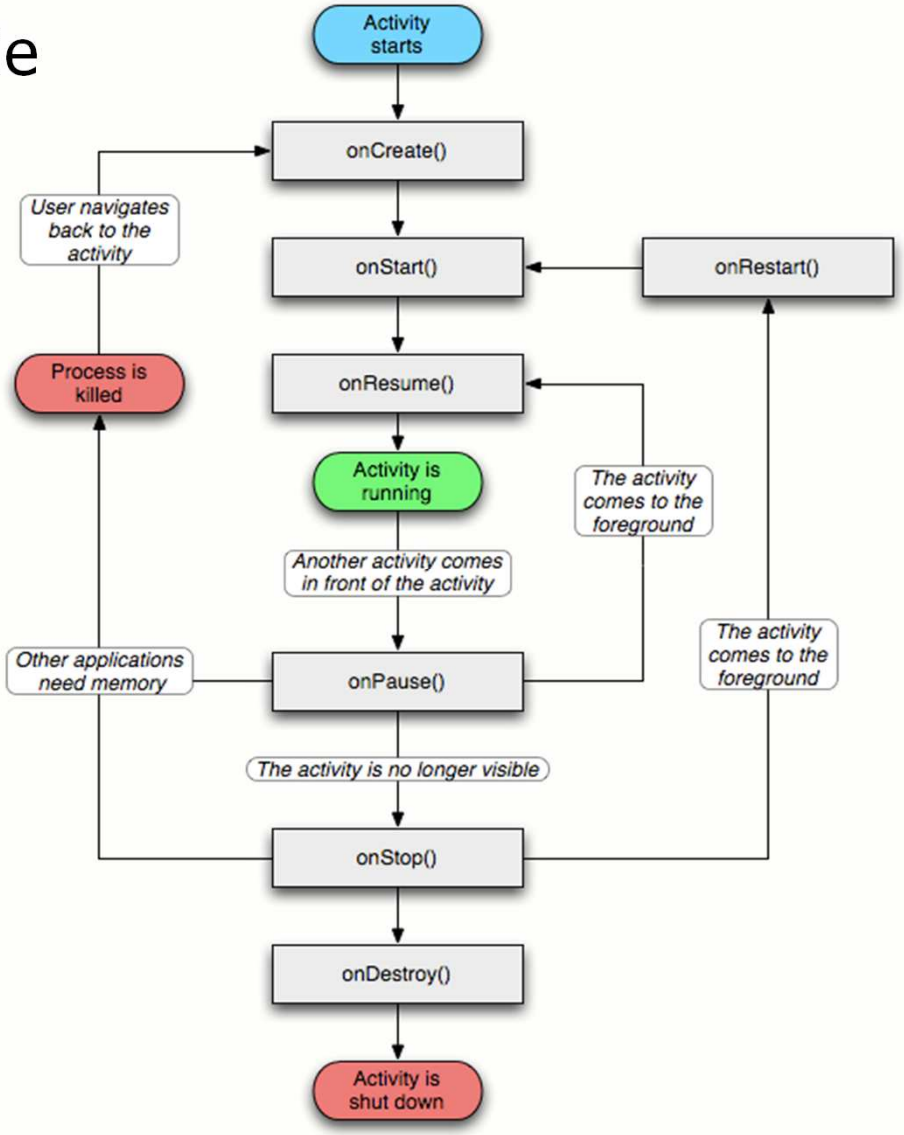
⇨ cycle de vie (1)

- Une activité n'a pas de contrôle direct sur son propre état.
- Son état dépend du système et des autres applications.
- A chaque changement d'état une méthode est appelée.
- Elles sont appelées suivant un ordre bien précis.
- On parle de ***cycle de vie*** de l'activité.

Les activités

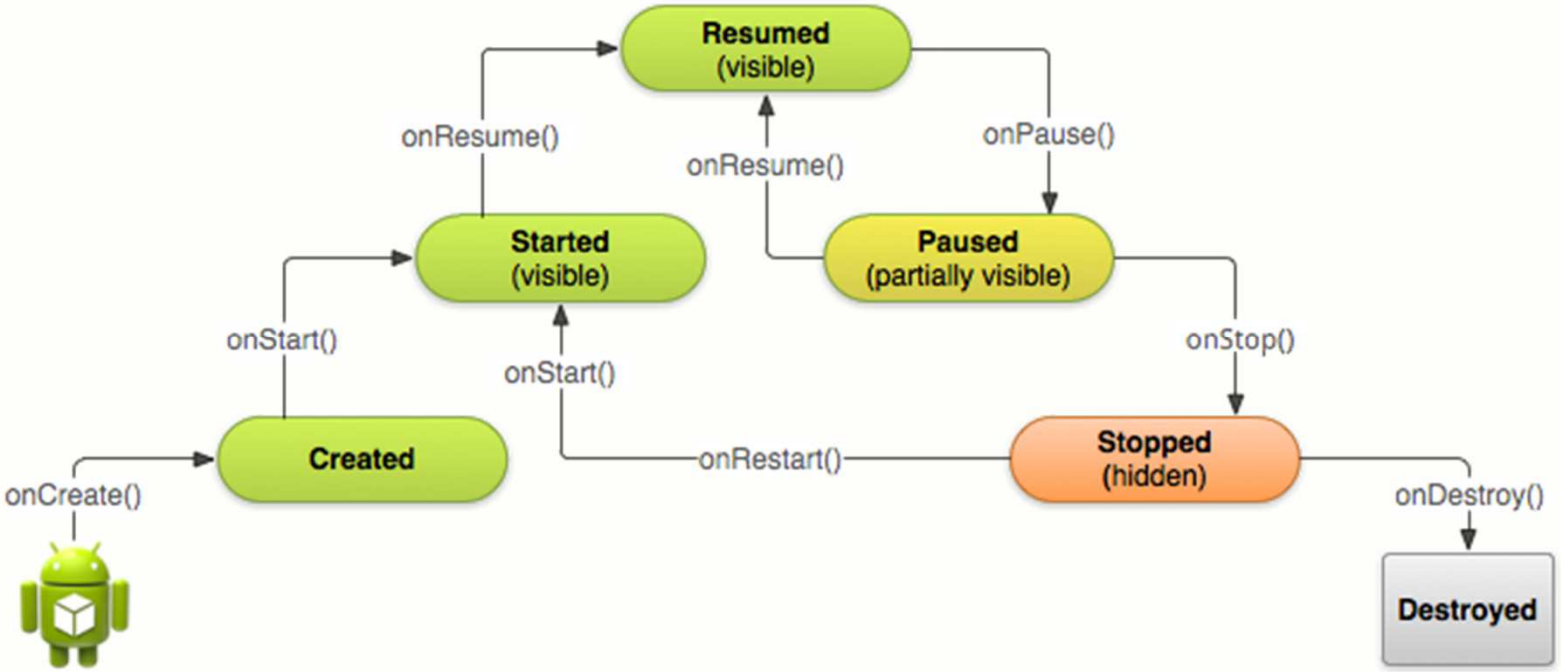
⇨ cycle de vie (2)

- cycle de vie



- L'automate (simplifié) du cycle de vie :

La *step pyramid*



Les activités

⇨ les états (1)

- Explication sur les états :
  - Les états *Created* et *Started* sont transient, on passe de l'un à l'autre après exécution des méthodes *onCreate()* puis *onStart()* puis *onResume()*
  - Les états *Resumed*, *Paused* et *Stoped* sont dits static (on peut y rester un certain temps)
  - Dans l'état *Paused*, l'activité est partiellement obscurcie par une autre. Il n'y a plus d'interaction avec l'utilisateur.
  - Dans l'état *Stoped*, l'activité est complètement masquée. L'objet existe toujours et garde ces valeurs.

Les activités

⇨ les états (2)

- Si l'utilisateur fait une action qui démarre une nouvelle activité, alors l'activité courante passe dans l'état *Stoped*.
- Si maintenant, l'utilisateur appui sur le bouton *back*, la première activité revient en passant par l'état *Started* puis *Resumed*. La deuxième activité passe dans l'état *Stoped* puis *Destroyed*.

➢ Le bouton *back* amène l'activité dans l'état Destroyed

➢ Un appel téléphonique amène l'activité dans l'état Stoped

Que dire des composants graphiques ?

- Le passage :

Resumed ⇨ Paused ⇨ Stopped ⇨ Started ⇨ Resumed

garde le contenu des composants graphiques

(pas besoin de sauvegarder une boite d'édition de texte)

- Même si le système détruit votre activité :

Le contenu des composants graphiques est sauvegardé dans un Bundle (un ensemble de paires clé/valeur)et les composants seront reconstruits a l'identiques

Destruction d'une activité

- Une activité peut être détruite lorsque :
  - L'utilisateur appuie sur le bouton back
  - L'application a lancé **finish()** sur l'activité
  - L'activité était dans l'état Stopped et Android a besoin de mémoire
- Une activité est détruite et recréée lorsque l'affichage passe du mode portrait au mode paysage.
  - Ceci est du au fait que l'on peut avoir des représentations différentes pour ces 2 modes.

Récupération après destruction

- Si l'activité a été détruite par Android (besoin de mémoire, contrainte système), les états des composants graphiques seront restitués.
- Si l'activité à été détruite par l'utilisateur, le contexte sera perdu et toutes les données par la même occasion.

Les ressources

⇨ introduction

- Toutes les ressources de l ’application vont se trouver dans des sous répertoires du répertoire « **res** »

  layout, menu, values, drawable …

- Les ressources autres que les images (layout, menu, texte, style, …) vont être fournis par des fichiers **xml**.
- Chaque ressource portera un nom.
- Les noms des sous répertoires de ressources peuvent avoir des qualificatifs et contenir le même nom de fichier xml (internationalisation).

  Ex : values-fr, values-en avec le fichier strings.xml

Les ressources

⇨ organisation

- *Les fichiers xml des ressources se trouvent dans des répertoire dont le nom est en rapport avec leurs utilisations.*
  - *drawable : images*
  - *Mipmap : icones de l'application*
  - *values : chaine de caractères, style, dimension*
  - *layout : les pages*
- *Tous ces répertoires peuvent être qualifiés afin de s'adapter au matériel utilisé.*
  - *la langue*
  - *la définition de l'écran*
  - *l'orientation de l'appareil*

Les ressources

⇨ qualificatifs (1)

Les qualificatifs seront ajoutés à la fin du nom et séparés par des moins « - »

- La syntaxe est :

  res/<type_de_ressource>[<-quantificateur 1> <-quantificateur 2>…<-quantificateur N>]

- Attention il y a une priorité à respecter
- Ils ne doivent pas être mis dans n'importe quel ordre.
- Il y en a 14 en tout mais seulement quelques uns sont utiles.
  - la langue
  - la taille de l'écran
  - l'orientation
  - la résolution

Les ressources

⇨ qualificatifs (2)

- la langue
  - priorité : 2
  - en : pour l'anglais ;
  - fr : pour le français ;
  - fr-rFR : pour le français utilisé en France ;
  - fr-rCA : pour le français utilisé au Québec ;
- la taille de l’écran (diagonale)
  - priorité : 3
  - small : petits écrans
  - normal : écrans standards ;
  - large : grands écrans (tablettes tactiles)
  - Xlarge : très grands écrans (téléviseurs)

Les ressources

⇨ qualificatifs (3)

- l'orientation
  - priorité : 5
  - port : pour *portrait*
  - land : pour *landscape (*paysage)
- la résolution
  - Priorité : 8
  - ldpi : environ 120 dpi ;
  - mdpi : environ 160 dpi ;
  - hdpi : environ 240 dpi ;
  - xhdpi : environ 320 dpi (> l'API 8) ;
  - nodpi : pour ne pas redimensionner les images matricielles (vous savez, JPEG, PNG et GIF !).

Les ressources

⇨ qualificatifs (4)

- Exemples :
  - res/drawable-small : pour des images spécifiquement pour les petits écrans.
  - res/drawable-large : pour des images spécifiquement pour les grands écrans.
  - res/layout-fr : pour avoir une mise en page sur un système français.
  - res/layout-fr-rFR : pour avoir une mise en page destinée à ceux qui ont choisi la langue *Française*
  - res/values-fr-rFR-port : pour des données qui s'afficheront uniquement à ceux qui ont choisi la langue *Française (France)* et dont le téléphone se trouve en orientation portrait.

Les ressources

⇨ la classe « R » (1)

- Dans les fichiers de ressources xml on utilise des noms pour distinguer les éléments.
- Dans un autre fichier xml, on peut récupérer une ressources par :

@typeRes/nomRes

- Dans du code java, on le fera par exemple pour une ressource de type chaine :

getRessources().getString(R.string.nomRes)

Les ressources

⇨ la classe « R » (2)

- Les composants seront eux repérés par des identificateurs.
- C'est le cas des composants graphiques.
- Dans un fichier xml, deux syntaxes sont possibles :
  - @+id/xxxx
  - @id/xxxx
- La première signifie qu'il faut créer cet id
- La deuxième fait référence à un id existant

- Mais où sont mémorisés ces noms de ressources et id ?

Les ressources

⇨ la classe « R » (3)

- En fait, Android génère automatiquement une classe « R » contenant une classe static pour chaque ressource qui elle-même contient des constantes entières correspondant aux ressources.

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
*
* This class was automatically generated by the
* aapt tool from the resource data it found.  It
* should not be modified by hand.
*/

package fr.afpa.cdi.mapremiereapplication;

public final class R {
   public static final class menu {
        public static final int menu_main=0x7f0d0000;
    }
    public static final class mipmap {
        public static final int ic_launcher=0x7f030000;
    }
    public static final class string {
        public static final int action_settings=0x7f0b0011;
        public static final int app_name=0x7f0b0012;
        public static final int hello_world=0x7f0b0013;
    }
}
```

Les ressources

⇨ la classe « R » (4)

Exemple de récupération d'un nom de ressource strings.xml (il y a un s)

- Il est accessible par:

  R.string.leNom (il n'y a pas de s)

- Dans le code java on le récupère par :

  getResources().getString(R.string.leNom)

- Dans un fichier xml :

  @string/leNom

Exemple de récupération d'un identifiant

- Dans le code java on récupère une vue par :

  findViewById(R.id.nomDeLaVue)

Deux possibilités sont à notre disposition :

- La construction procédurale (~ swing)

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

        TextView tv = new TextView(this);
        tv.setText("Coucou !");

        setContentView(tv);
}
```

- La construction déclarative (à utiliser)

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

        setContentView(R.layout.activity_main);
}
```

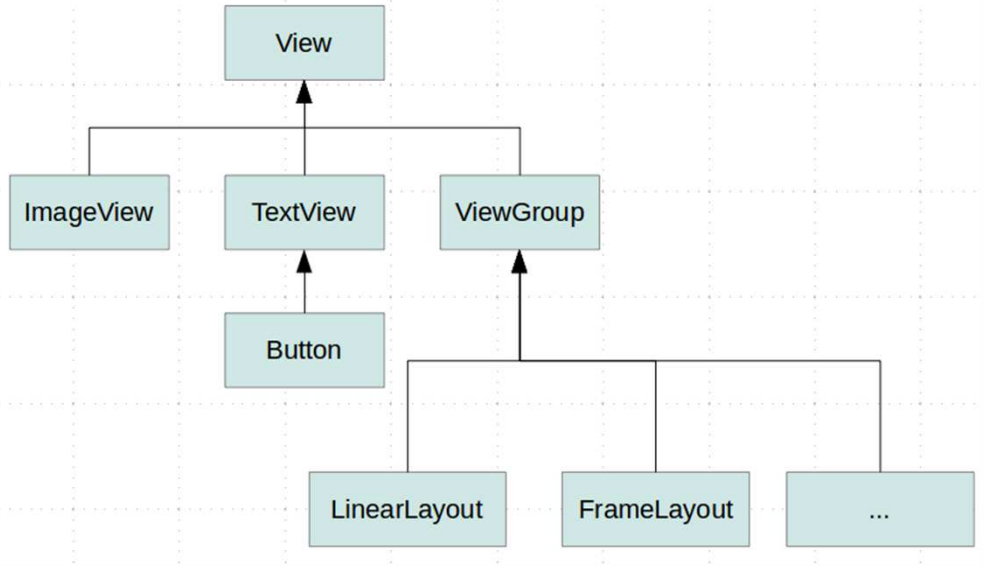
fichier : activity_main.xml
dans res\layout

```
<linearLayout>
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="coucou" />
</linearLayout>
```

Les interfaces graphiques

⇨ la classe View et ViewGroup

- La classe de base est la classe VIEW
  - Import : android.view.View
- La classe de base des conteneurs est ViewGroup
  - Import : android.view.ViewGroup
- ViewGroup hérite de View



Les interfaces graphiques

⇨ les principales Views

- Il existe un grand nombre de Views
- Les principales sont :
  - TextView : label
  - EditText : champ de saisie
  - Button : bouton cliquable
  - RadioButton : les boutons radio
  - CheckBox : case à cocher
  - Spinner : boite de sélection
  - ImageView : conteneur d'image
  - DatePicker : sélecteur de date
  - Toast : message pop-up

Les Views

⇨ le TextView (1)

- C'est un affichage : label
- Quelques attributs :
  - android:layout_width
  - android:layout_height

  } wrap_content, match_parent, xxdp
  - android:text — Le texte à afficher
  - android:typeface — La police
  - android:textstyle — Italic, bold_italic, …
  - android:textcolor — La couleur RGB en hexa : #FF0000

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

- Dans le code java, une grande partie des setters existent et correspondent aux attributs du xml.
- Exemple :
- setText(" … ") ⟺ android:text=" … "
- setLines(nb) ⟺ android:lines="nb"

- Un seul moyen pour les connaître :

La documentation

- C'est une boite de saisie.
- Attention à la récupération de la saisie utilisateur :

```
EditText edtNom = (EditText)findViewById(R.id.editText_nom);
String nom = edtNom.getText().toString();
```

- La méthode *getText()* sur un objet *EditText* retourne un *Editable*.
- Il faut employer *toString()* pour récupérer la saisie faite par l'utilisateur.

- Un bouton peut avoir un texte ou une icone voir les deux :

texte :

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

icone :

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

les deux :

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

La gestion événementielle

⇨ introduction

- Deux possibilités
  - Faite par code par l'implémentation d'écouteurs.
  - Faite en ajoutant un attribut à la view dans le fichier xml.
- On préfèrera la première même si la deuxième semble plus simple.
- Certain composant sont directement auditrices d'événements.

La gestion événementielle

⇨ créer un écouteur (1)

- Chaque View possède des méthodes setOnXXXListener() permettant d'enregistrer un écouteur.
- La création d'un écouteur se fait en instanciant des interfaces de nom OnXXXListener.
- Ce qui oblige à implémenter des méthodes de nom onXXX().
  - onClick(), onLongClick(), onKey(), …

La gestion événementielle

⇨ créer un écouteur (2)

- Exemple de Click sur un bouton

```
@override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.monecran);

    // On “implémente ??” l’interface OnClickListener
    OnClickListener clickButton = new OnClickListener()
    {
        @override
        public void onClick(View actuelView)
        {
            // traitement déclenché par le clic sur le bouton
            // par exemple affichage d’une boite popup
            Toast.makeText(getBaseContext(), "Message", Toast.LENGTH_LONG).show();
        }
    };

    // On récupère le bouton cliqué
    Button bouton = (Button) findViewById(R.id.createAcount);

    // On appelle sa méthode setOnClickListener()
    bouton.setOnClickListener(clickButton);
}
```

La gestion événementielle

⇨ créer un écouteur (3)

- Par l'attribut android:onClick
  - Dans le fichier xml de description de la page :

```
<Button
        android:id="@+id/btnOk"
        android:onClick="onClickTraiter"
        android:text="@string/btn_ok" />
```

  - Dans le code de l'activité :

```
public void onClickTraiter(View v) {
        // traitement
}
```

Les Views

⇨ bouton radio (1)

- A regrouper dans un « RadioGroup »
- Un *RadioGroup* étend la classe *LinearLayout* et par défaut à une orientation verticale.
- Pour trouver le bouton radio sélectionné utiliser :

  getCheckedRadioButtonId() sur le RadioGroup
- Pour détecter un changement utiliser :

  setOnCheckedChangeListener(
  RadioGroup.onCheckedChangeListener listener)

  sur le RadioGroup et redéfinir

  onCheckedChanged(RadioGroup rdgr,
  int rdCheckedId)

- Exemple pour un groupe de 2 boutons :

```
RadioGroup rdG1 = (RadioGroup)findViewById(R.id.radioGroup1l);

rdG1.setOnCheckedChangeListener(new
                                RadioGroup.OnCheckedChangeListener(){

    @Override
    public void onCheckedChanged(RadioGroup rdgr, int rdCheckedId) {

        if (rdCheckedId == R.id.radioButton1){
            // traitement bouton 1;
        } else {
            // traitement bouton 2;
        }
    }
});
```

- Autre possibilité en utilisant un attribut dans le xml de définition du layout :

```
<RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="horizontal">
   <RadioButton android:id="@+id/radio1"
      android:text="@string/rd1"
      android:onClick="onRadioButtonClicked"/>
   <RadioButton android:id="@+id/radio2"
      android:text="@string/rd2"
      android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

```
public void onRadioButtonClicked(View view) {
      switch(view.getId()) {
      case R.id.radio1:
         // Pirates are the best
         break;
      case R.id.radio2:
         // Ninjas rule
         break;
   }}
```

Les interfaces graphiques

⇨ les Layouts

- Afin de mettre en page tous les composants graphiques on fait appel à la notion de *Layout*.
- Un Layout est un conteneur (ViewGroup)
- Ils sont représentés par une classe

  XXXLayout
- Les principaux sont :
  - LinearLayout
  - RelativeLayout
  - GridLayout
- La mise en page de chaque activité sera décrite dans un fichier Xml dont la racine portera le nom du layout choisi.

Les conteneurs de disposition

⇨ LinearLayout (1)

- Les composants (View) sont rangés les uns à la suite des autres soit horizontalement, soit verticalement.
- L'orientation est définie dans le fichier Xml de l'activité en ajoutant l'attribut *android:orientation* à la balise *linearLayout*
  - valeurs possibles : *vertical*, *horizontal*
- Il est possible de changer l'orientation par programmation en utilisant la méthode *setOrientation(valeur)* de l'objet *LinearLayout*
  - « valeur » vaut *LinearLayout.HORIZONTAL* ou *LinearLayout.VERTICAL*

Les conteneurs de disposition

⇨ LinearLayout (2)

- Exemple de fichier : activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:text="@string/invite" />

</LinearLayout>
```

Les conteneurs de disposition

⇨ RelativeLayout (1)

- Les composants sont rangés les uns par rapport aux autres.
- C'est le layout par défaut d'Android Studio.
- Un composant indique sa position
  - *soit par rapport à son parent*
  - *soit par rapport à ses frères*
- *Chaque composant possèdera un id (voir plus loin) pour le repérer.*
- *Les valeurs d'attributs seront soit des booléen soit des id.*

Les conteneurs de disposition

⇨ RelativeLayout (2)

- Exemple de fichier : activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >
<TextView
        android:text="@string/creerPel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="25sp"
        android:id="@+id/txtTitre" />
    <TextView
        android:id="@+id/txtView1"
        android:layout_below="@+id/txtTitre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
       android:text="@string/hello_world" />
</RelativeLayout>
```

Les conteneurs de disposition

⇨ RelativeLayout (3)

Quelques attributs possibles :

- android:layout_alignParentTop :

  Si true la vue est calée sur le haut de la vue parente
- android:layout_centerVertical :

  Si true la vue est centrée verticalement dans son parent
- android:layout_below :

  Le haut de la vue est en dessous de la vue indiquée (par son id)
- android:layout_toRightOf :

  Le coté gauche de la vue est à droite de la vue indiquée (par son id)

Enchaîner les activités

⇨ les Intents (1)

- Pour passer d'un écran à l'autre (d'une activité à une autre activité de la même application ou pas) il faut lancer une intention.
- Une intention est une instance de la classe **Intent**.
- Deux types d'intention :
  - implicite
    - Android décidera l'application qui répondra à cette intention
  - explicite
    - vous demanderez un service ou une activité en utilisant leur nom de classe

Enchaîner les activités

⇨ les Intents (2)

L'intention explicite :

- Pour passer d'un écran à l'autre (d'une activité à l'autre) il faut lancer une intention.
  - Création de l'objet Intent :

```
Intent nomInt = new Intent(Context appContext, Class<?> cls);
```

  - Lancement de l'intention

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

  - Exemple :

```
Intent monIntent = new Intent(getApplicationContext(),
                              NouvelleActivity.class);
startActivity( monIntent );
```

L'intention implicite :

- Exemple d'un appel téléphonique

```
Intent intent = new Intent(Intent.ACTION_DIAL,
                                Uri.parse(tel:01-02-03-04-05));

PackageManager pm = getPackageManager();

ComponentName cn = intent.resolveActivity(pm);

if(cn == null){
        /*toDo pas d'application*/
} else {
        startActivity(intent);
}
```

Enchaîner les activités

⇨ les Intents (4)

Passage de données entre activités.

- Un Intent peut servir de conteneur pour passer des informations à l'activité.
- Il faut avant de la lancer, appeler la méthode putExtra() :

```
public Intent putExtra(String nomDeExtra, valeur_typée);
```

- Dans cette nouvelle activité, on récupère l'Intent qui l'a lancée par getIntent().
- On appelle la méthode getExtras() sur cet Intent, puis on appelle une méthode correspondant au type de donnée :
- getXXX(nomDeExtra, valeurParDéfaut)

Enchaîner les activités

⇨ les Intents (5)

- Exemple :
  - Dans l'activité appelante :

```
// On crée l'Intent qui va nous permettre d'afficher l'autre Activity
Intent intent = new Intent(this, ActivitySuivante.class);

// On affecte à l'Intent les données à passer
intent.putExtra("param1",valeur1);          // valeur1 est une String
intent.putExtra("param2",valeur2);          // valeur2 est une String
...
 // On démarre l'autre Activity
startActivity(intent)
```

  - Dans l'activité appelée :

```
// On récupère l'Intent que l'on a reçu
Intent thisIntent = getIntent();

// On récupère les paramètres passés
String par1 = thisIntent.getExtras().getString("param1");
String par2 = thisIntent.getExtras().getString("param2");
...
```

Enchaîner les activités

⇨ les Intents (6)

L'activité appelée doit retourner des données:

- Dans l'activité appelante :
  - faire appel à startActivityForResult(Intent intent, int requestCode)
    - requestCode est un identificateur d'appel
- Dans l'activité appelée :
  - créer un Intent en lui ajoutant les données par putExtra()
  - faire appel à setResult(int, Intent)
    - l'int est un code de retour qui peut prendre RESULT_CANCELED, RESULT_OK ou autre

Enchaîner les activités

⇨ les Intents (7)

L'activité appelante récupère les données:

- Dans l'activité appelante, la méthode suivante sera automatiquement appelée :

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

  - le requestCode est celui que vous avez donné lors du startActivityForResult()
  - le resultCode est celui du setResult()
  - data est intent de retour

Les interactions de l'utilisateur

⇨ message Toast

- Pour avertir l'utilisateur, on peut afficher un message temporaire. Le *Toast*
- Pour créer un Toast il faut appeler la méthode static makeText() de la classe Toast puis l'afficher par sa méthode show().

```
public static Toast makeText(Context context,
                             CharSeqence text,
                             int durée)
```

- Exemple :

```
Toast monToast = Toast.makeText(getBaseContext(),
                                "message",
                                Toast.LENGTH_SHORT);
monToast.show();
```

```
Toast.makeText(getBaseContext(),"message",
                                Toast.LENGTH_LONG).show();
```

Les interactions de l'utilisateur

⇨ les message Snackbar

- Le Snackbar remplace les Toast
- Pour créer un Snackbar appeler la méthode static de la classe Snackbar :

```
Snackbar mySnackbar = Snackbar.make(viewId, stringId, duration);
```

- Pour l'afficher appeler la méthode show() :

```
mySnackbar.show()
```

- Pour un simple message on peut chainer :

```
Snackbar.make(findViewById(R.id.myCoordinatorLayout),
        R.string.email_sent,
            Snackbar.LENGTH_SHORT)
        .show();
```

- Il est possible de lui associer un gestionnaire d'évènement par la méthode :

```
setAction(stringId, new MyListener())
```

Interaction avec l'utilisateur

⇨ les menus (1)

- Plusieurs types de menus peuvent être créés.
  - menu de la barre d'action
  - menu contextuel
  - menu personnalisé
- On peut les créer de deux manières différentes :
  - par du code java
  - par fichier xml
- Nous nous intéresserons au premier type de menu créé en xml.

- Description du menu dans un fichier xml :

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

   <item android:id="@+id/option"
      android:title="Options"
      android:icon="@drawable/option">

      <menu android:id="@+id/sousmenu">
         <item android:id="@+id/item1"
            android:title="Item1"   />

         <item android:id="@+id/item2"
            android:title="Item2"  />
      </menu>
   </item>

   <item android:id="@+id/quitter"
      android:title="Quitter"
        android:icon="@drawable/quit" />
</menu>
```

Interaction avec l'utilisateur

⇨ les menus (3)

- Dans le code de l'activité :
  - L'activité doit étendre ActionBarActivity.
  - La méthode onCreateOptionsMenu() doit être redéfinie.

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    MenuInflater inflater = getMenuInflater();
    // Instanciation du menu XML spécifier en un objet Menu
    inflater.inflate(R.menu.nom_menu, menu);
    return true;
}
```

  - On récupère un MenuInflater qui par sa méthode inflate() et le fichier de définition xml va créer un objet de la classe Menu
  - Il faut obligatoirement retourner True pour que le menu soit affiché.

Interaction avec l'utilisateur

⇨ les menus (4)

- Toujours dans le code de l'activité :
  - La méthode onOptionsItemSelected() doit être redéfinie.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // On teste l'Id de l'item cliqué et on déclenche une action
    switch (item.getItemId()) {
        case R.id.item1:
            // action 1
            return true;
        case R.id.item21:
            // action 2
            return true;
        case R.id.item3:
            // action 3
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Utilisation de la méthode getItemId() sur l'objet MenuItem passé en paramètre permet de connaître l'id de l'item qui a été cliqué.

Interaction avec l'utilisateur

⇨ les menus contextuels (1)

- Un composant graphique peut être à l'origine de l'apparition d'un menu lorsque l'utilisateur fait un appui prolongé sur celui-ci.
- Dans l'activité, il faut faire appel à

  registerForContextMenu(leComposantGraphique)

  afin que ce composant gère l'événement.
- Lors de cet événement, la méthode

  public void onCreateContextMenu(ContextMenu menu,View v, ContextMenuInfo menuInfo)
- Est lancée.
  - menu : l'objet menu qui sera construit
  - v : le composant graphique
  - menuInfo : infos supplémentaires

Interaction avec l'utilisateur

⇨ les menus contextuels (3)

- Lorsqu'un item du menu contextuel est sélectionné, l'activité appelle la méthode :

```
public boolean onContextItemSelected(MenuItem item)
```

- La méthode *getItemId()* lancée sur l'argument *item* permet de connaître l'item sélectionné.

- Le traitement reste le même que celui d'un menu normal.

Interaction avec l'utilisateur

⇨ la barre d'actions

- Il suffit d'utiliser l'option « showAsAction » sur l'Item dans le fichier de description du menu :

app:showAsAction="*ifRoom*"

- et d'utiliser :

android:icon="*@drawable/mon_icone*"

si on préfère utiliser une icône :

Fin de la deuxième partie