Chapitre 3

Les interfaces utilisateurs avec Android

Plan du chapitre 3

- IHM des smartphones, IHM pour Android
- Les deux principes des IHM
- Un second programme : IHM par programmation, par description
- Les `Layout`, les contrôles
- La gestion des événements
- Enchaîner les écrans
- `Toast` et traces
- L'internationalisation
- Les icônes, menus, notifications et boîtes de dialogues
- Le multi-plateformes

Smartphone != ordinateur

- Android tire partie des particularités des smartphones :
  - interface homme machine adapté (tactile, multitouch)
  - divers modes : vibreur, sonnerie, silencieux, alarme
  - notifications (d'applications, d'emails, de SMS, d'appels en instance)
  - de boussole, accéléromètre, GPS
  - divers capteurs (gyroscope, gravité, baromètre)
  - NFC, RFID
  - téléphonie (GSM) et réseau EDGE, 3G, 4G, SMS, MMS
  - Appareil photo, caméra vidéo (enregistrement et rendu)
  - une base de données intégrée (SQLite)
- En plus de ce qu'on peut avoir sur un ordinateur : navigateur, bibliothèques graphiques 2D, 3D (Open GL), applications de rendu multimédia (audio, vidéo, image) de divers formats, réseau Bluetooth et Wi-Fi, caméra

Les IHM graphiques avec Android

- Bibliothèque propre
- Pas AWT, ni Swing, ni Java ME / LCDUI
- Décrit par fichier XML
- Ecran en Android géré par une activité

Activité (Activity)

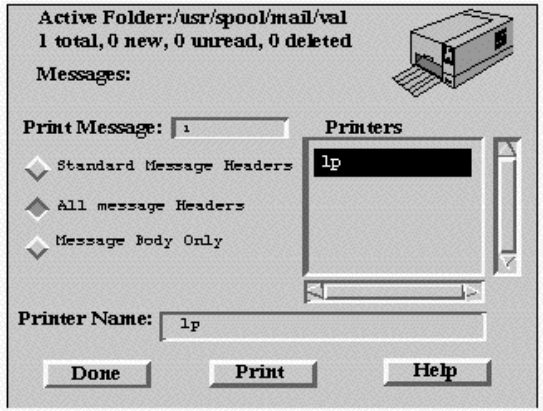
- Correspond à une seule classe Java
- Une activité gère l'affichage et l'interaction d'un écran (IHM)
- Gère les événements, lance l'affichage d'autres écrans, lance du code applicatif
- Suit un cycle de vie déterminé (cf. applets, midlets, servlets, EJB, ...)
- Utilise les `Intent` pour lancer d'autres activités

Construction d'une IHM

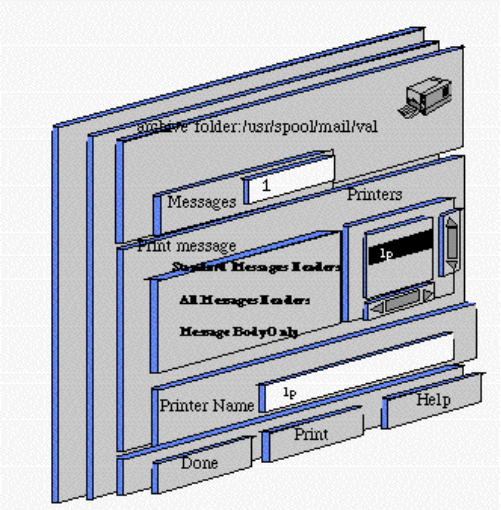
- Plutôt en XML mais
- XML ne peut pas être débogué !
- Tout ne peut pas être fait en XML

Premier principe des IHM (1/4)

- Quand on voit ceci :

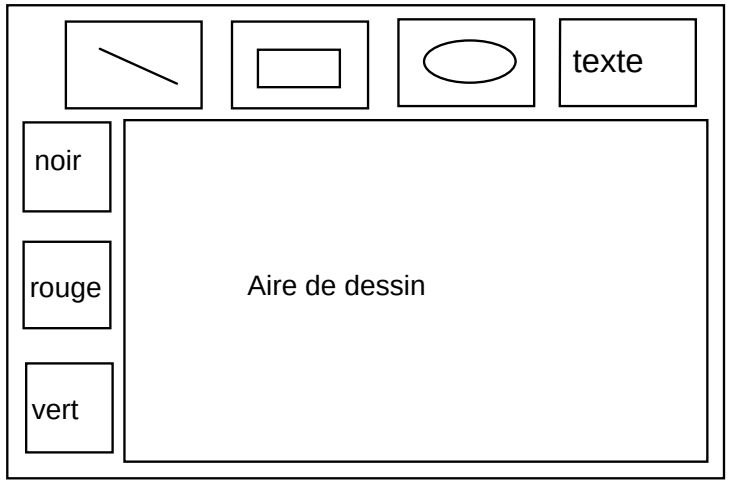


- C'est qu'on a programmé cela :

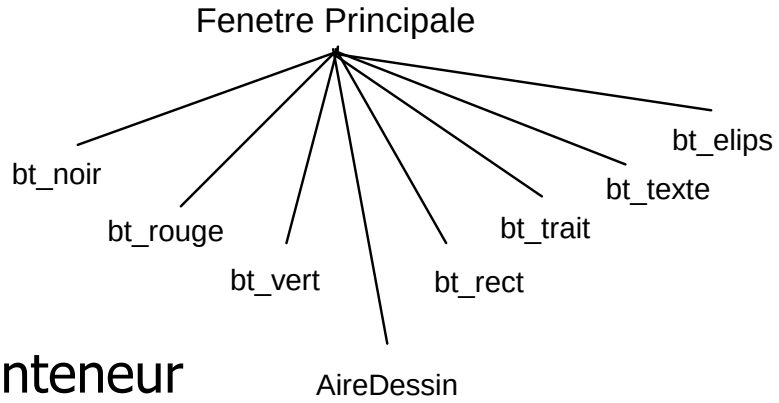


Premier principe des IHM (2/4)

- Si on veut :



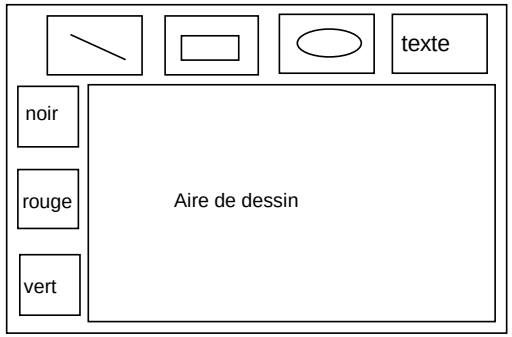
- On doit construire cela :



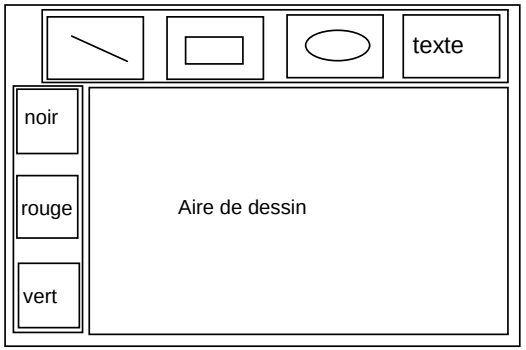
- => Fenetre principale = un conteneur

Premier principe des IHM (3/4)

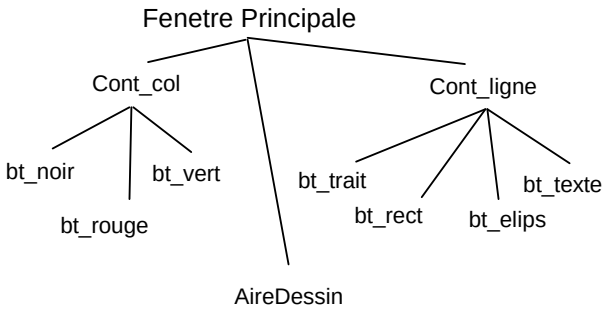
- Plus sûrement, si on veut :



on écrit plutôt :



c'est à dire :



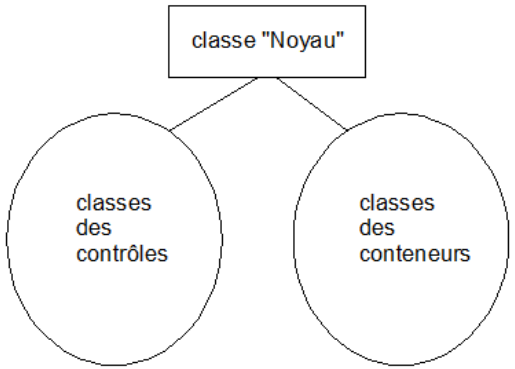
Premier principe des IHM (4/4)

- (Premier principe) : construire une IHM, c'est mettre des composants graphiques les uns à l'intérieur des autres
- Il y a donc, dans une IHM à présenter à l'utilisateur, un arbre de composants graphiques
- Les éléments de cet arbre sont des composants graphiques (redite !)
- Etre "fils de" dans cet arbre signifie "être contenu dans"

- Voilà pour les composants graphiques ! (et le premier principe des IHM)

Second principe des IHM (1/3)

- Les ensembles de composants graphiques sont des classes. On aura la classe des boutons, la classe des cases à cocher, etc.
- Un composant graphique particulier sera une instance particulière d'une classe. Par exemple le bouton "Quitter" et le bouton "Sauvegarder" d'une IHM seront deux instances de la classe des boutons : merci l'OO !
- Il y a une famille de conteneurs et une famille de non conteneurs
- D'où les classes de composants graphiques :

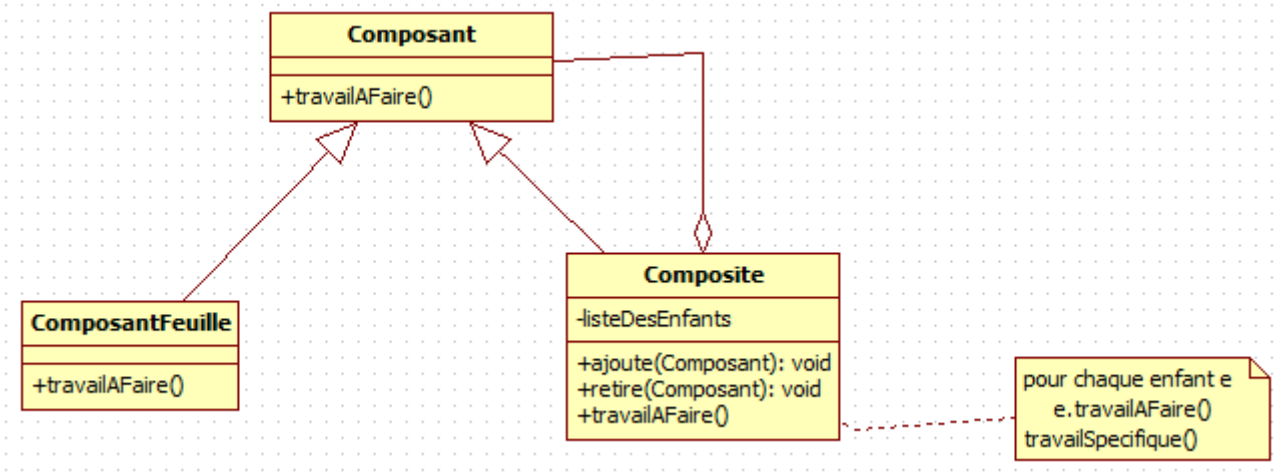


- Question : Comment sont rangées ces classes ?

- Réponse : dans un arbre d'héritage de classes : merci l'OO (bis) !

Second principe des IHM (2/3)

- Plus précisément le point de départ de l'arborescence des classes est :



- C'est le design pattern Composite
- remarque : `travailAFaire()` est répercuté sur tout l'arbre des instances

Second principe des IHM (3/3)

- (Second principe) : les bibliothèques pour construire des IHM sont, en Java, (et souvent !) des classes rangées par arborescence d'héritage
- Les éléments de cette arborescence sont des classes (redite !)
- Etre "fils de" dans cette arborescence signifie "hérite de"
- Voilà pour les classes (et le second principe des IHM)

Les deux principes des IHM

- Lorsqu'on parle d'IHM, il existe deux arborescences et donc deux "principes"
- 1er principe : Une IHM est construite en mettant des composants graphiques les uns à l'intérieur des autres
- 2ième principe : les composants graphiques sont obtenus comme instances de classes. Ces classes sont rangées dans une arborescence d'héritage
- Remarque : Ces deux arbres (celui des composants graphiques et celui des classes) ont peu de choses à voir l'un l'autre
  - Le premier est l'architecture de l'interface i.e. le placement des divers composants graphiques les uns par rapport aux autres,
  - le second est un arbre d'héritage de classes donné une bonne fois par le concepteur de la bibliothèque graphique

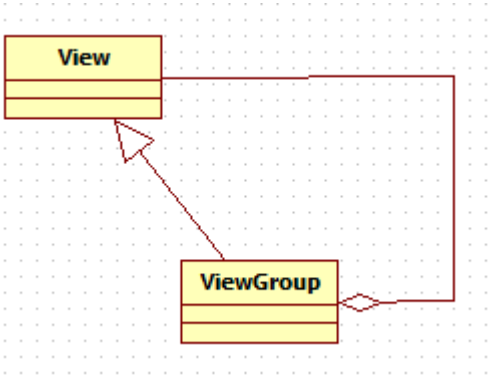
Un composant IHM = 3 parties

- Un composant graphique a 3 parties :
  - les données qu'il représente : le modèle (model)
  - le dessin d'affichage : la vue (view)
  - ce qui prend en charge les actions de l'utilisateur sur ce composant : le controleur (controler)
- C'est l'architecture MVC
- "L'idée est de bien séparer les données, la présentation et les traitements" (*)
- Les traitements sont souvent délégués à un objet autre que le composant graphique lui même (et qui n'est pas, en général, un composant graphique) : programmation par délégation

- (*) source : `http://fr.wikipedia.org/wiki/Paradigme_MVC`

Les fondamentaux d'Android

- La classe "Noyau" de base est la classe `android.view.View` (~ `java.awt.Component` de AWT)
- La classe de base des conteneurs est `android.view.ViewGroup` (~ `java.awt.Container` de AWT)
- On a donc :



- En Android, les conteneurs sont souvent appelés les Layout, les contrôles sont parfois appelés des widgets (window objects)

IHM : construction procédurale vs. déclarative

- En Android on peut construire les IHM en codant en Java des instructions ...
- ... ou en décrivant l'IHM par un fichier XML
- La première solution est celle habituelle de Java (SE Swing et AWT, ME) ou d'autres domaines (Motif, Openwin de Sun, ...)
- La seconde est courante dans certains domaines (Apple, ...)

- Une nouvelle : on peut construire la plupart des IHM en glisser-déposer cf. Interface Builder de NeXT : Jean-Marie Hullot (1989) , Visual Basic, ...

Un second programme

- On veut créer une application Android qui affiche un bouton. Lorsque l'utilisateur actionne le bouton, l'heure est affichée. Le résultat est :

- L'heure est mise à jour lorsqu'on actionne le bouton

Code du second programme (1/2)

- On peut tout coder en Java dans l'activité :

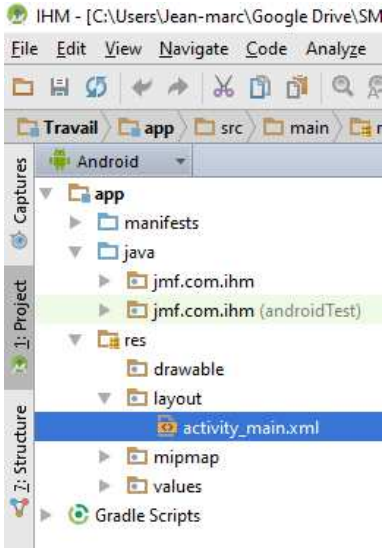
```
package android.jmf;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import java.util.Date;
public class BoutonHeureActivite extends Activity implements View.OnClickListener
{
    private Button btn;
    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        btn = new Button(this);
        btn.setOnClickListener(this);
        updateTime();
        setContentView(btn);
    }
    public void onClick(View view) {
        updateTime();
    }
    private void updateTime() {
        btn.setText(new Date().toString());
    }
}
```

Code du second programme (2/2)

- L'activité est le listener du bouton :
  `public class BoutonHeureActivite extends Activity implements View.OnClickListener`
- Ce qui nécessite d'implémenter la méthode :
  `public void onClick(View view)` qui est lancée lorsqu'on actionne le bouton (technique des listeners cf. événement Java SE)
- Le bouton est mis dans l'activité (`setContentView(btn)`)

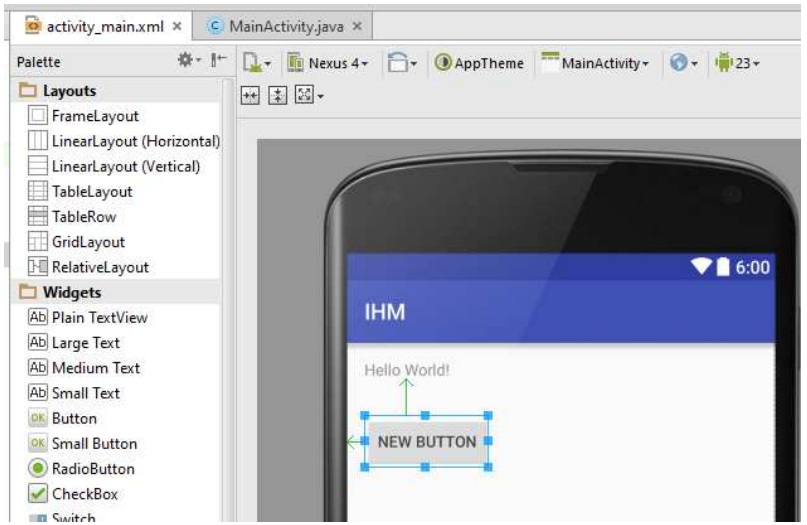
Second programme : une autre solution (1/5)

- En général, pour l'IHM on utilise plutôt les fichiers XML
- Par exemple, créer une nouvelle application Android
- Ouvrir le fichier `activity_main.xml` (dans `res/layout/activity_main.xml`)
- Et on peut construire l'IHM en glisser déposer !



Second programme : une autre solution (2/5)

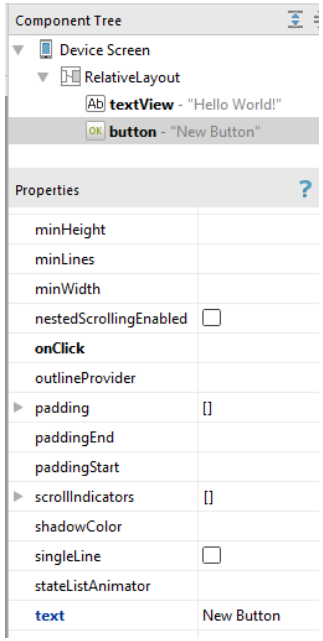
- La fenêtre de `activity_main.xml` propose deux onglets : l'onglet Design et l'onglet Text
- L'onglet Text donne le code XML de ce fichier
- L'onglet Design (qui met parfois du temps à s'afficher lors de sa première ouverture) permet de visualiser l'IHM correspondant à ce fichier (partie droite) et de construire cette IHM en glisser-déposer avec les éléments graphiques de la partie gauche



- Par exemple, on peut sélectionner le composant `Button` et l'amener dans la partie droite
- Ce constructeur d'IHM est l'ADT Visual Designer

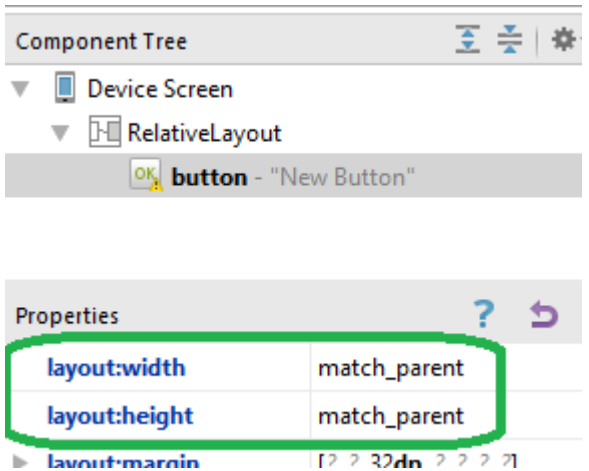
Second programme : une autre solution (3/5)

- L'onglet Component Tree indique l'arborescence des composants graphiques de l'application
- Lorsqu'on sélectionne le composant graphique (quelle que soit la fenêtre !) apparaît l'onglet Properties : ce sont les propriétés du composant graphique



Second programme : une autre solution (4/5)

- On peut changer la largeur et la hauteur d'un composant à l'aide des propriétés `layout:width` et `layout:height`



- Les valeurs de ces propriétés peuvent être
  - `match_parent` (anciennement nommé `fill_parent` avant l'API 8) indique que le composant graphique sera aussi grand en largeur ou hauteur que son conteneur le lui autorise
  - `wrap_content` indiquant que le composant prend seulement la taille qui lui faut en largeur ou hauteur pour s'afficher correctement

- source : `http://developer.android.com/guide/topics/ui/declaring-layout.html`

Second programme : une autre solution (5/5)

- L'IHM est alors généré cf. les deux onglets de `activity_main.xml` :

activity_main.xml × | app × | AndroidManifest.xml ×

```
<?xml version="1.0" encoding="utf-8"?>
<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" android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp" tools:context=".MainActivity">

    <Button
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="New Button"
        android:id="@+id/button"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="32dp"
        android:layout_alignParentEnd="false" />
</RelativeLayout>
```

Correspondance attribut - méthode

- Pour positionner une propriété (= valeur d'attribut = données = ...) d'un composant graphique on peut le faire soit en XML soit par appel d'une méthode appropriée
- Dans le fichier XML, on positionne une propriété d'un composant graphique à l'aide d'une valeur d'attribut de sa balise XML
- Donc il y a une correspondance entre les noms d'attribut d'une balise XML et une méthode, pour positionner une propriété d'un composant graphique. On a, par exemple :

XML Attributes		
Attribute Name	*Related Method*	*Description*
android:alpha	setAlpha(float)	alpha property of the view, as a value between 0 (completely transparent) and 1 (completely opaque).
android:background	setBackgroundResource(int)	A drawable to use as the background.
android:clickable	setClickable(boolean)	Defines whether this view reacts to click events.
android:contentDescription	setContentDescription(CharSequence)	Defines text that briefly describes content of the view.
android:drawingCacheQuality	setDrawingCacheQuality(int)	Defines the quality of translucent drawing caches.
android:duplicateParentState		When this attribute is set to true, the view gets its drawable state (focused, pressed, etc.) from its direct parent rather than from itself.
android:fadeScrollbars	setScrollbarFadingEnabled(boolean)	Defines whether to fade out scrollbars when they are not in use.
android:fadingEdgeLength	getVerticalFadingEdgeLength()	Defines the length of the fading edges.
android:filterTouchesWhenObscured	setFilterTouchesWhenObscured(boolean)	Specifies whether to filter touches when the view's window is obscured by

à `http://developer.android.com/reference/android/view/View.html`

Identifier les composants = la méthode "miracle"

- Le fichier `activity_main.xml` repère les composants par `android:id`

```
<Button
    android:id="@+id/button1"
    android:layout_width="fill_parent"
```

- Dans cet exemple il s'agit de `button1`
- Le composant est manipulé par cet identifiant dans le programme Java à l'aide de la méthode ("miracle")
  `findViewById(R.id.`*`nomIdentifiant`*`);`
- La valeur *`nomIdentifiant`* est celle qui apparaît dans le fichier `activity_main.xml` après `@+id/`
- Pour cet exemple ce sera `findViewById(R.id.button1);`

Le traitement de `setContentView()`

- `setContentView()` n'est pas banale ! Elle est capable de lire un fichier xml, l'analyser, construire des objets Java (souvent des `View`) à partir de ces données, les agencer pour construire une IHM
- Ce mécanisme est dit inflate

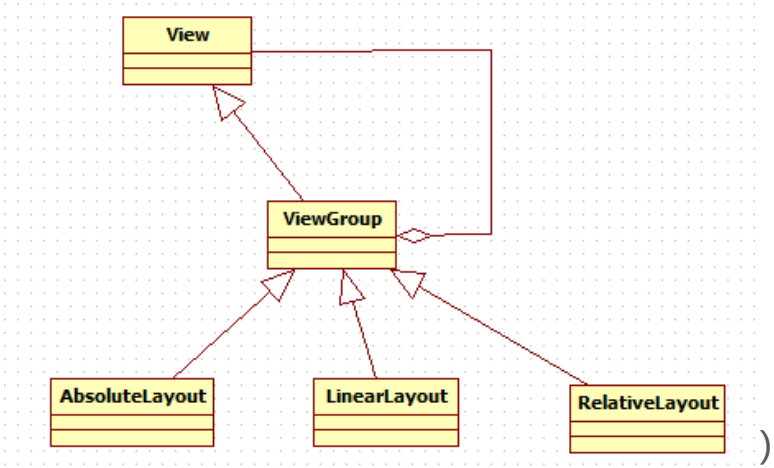
Second programme, solution 2 : le code

- Comme toute l'IHM est faite dans `activity_main.xml`, voici le code de l'activité :

```
package android.jmf;

import java.util.Date;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class BoutonHeure2Activite extends Activity implements View.OnClickListener {
     private Button btn;
     @Override
     public void onCreate(Bundle icicle) {
          super.onCreate(icicle);
          setContentView(R.layout.activity_main);
          btn=(Button)findViewById(R.id.button1);
          btn.setOnClickListener(this);
          updateTime();
     }
     public void onClick(View view) {
          updateTime();
     }
     private void updateTime() {
          btn.setText(new Date().toString());
     }
}
```

Les Layout

- Les conteneurs Android sont souvent des `XXXLayout` !
- C'est un peu différent de Java AWT ou Swing. Un Layout Android est un container et un Layout AWT à la fois
- Les principaux Layout Android sont :
  - `LinearLayout` (~ un containeur AWT géré par un FlowLayout AWT)
  - `RelativeLayout`
  - `AbsoluteLayout` (déprécié depuis l'API 3 !)
- On a donc :



)

LinearLayout

- Les composants à l'intérieur d'un `LinearLayout` sont rangés les uns à la suite des autres horizontalement ou verticalement
- Principales propriétés d'un `LinearLayout` : l'orientation, le mode de remplissage (fill model)

Orientation d'un `LinearLayout`

- L'orientation indique si le `LinearLayout` présente ces contenus sur une ligne (horizontalement) ou sur une colonne (verticalement)
- La propriété d'orientation à utiliser pour un `LinearLayout` dans le fichier XML est l'attribut `android:orientation` de la balise `LinearLayout`. Les valeurs possibles pour cette propriété sont `vertical` et `horizontal`
- La valeur `vertical` indique que les contenus seront les uns en dessous des autres, la valeur `horizontal` indique qu'ils seront les uns à la suite des autres
- L'orientation peut être modifiée à l'éxécution par la méthode `setOrientation()` lancée sur le `LinearLayout` en précisant la valeur `LinearLayout.HORIZONTAL` ou `LinearLayout.VERTICAL`

Mode de remplissage (fill model) d'un `LinearLayout`

- Cela concerne les attributs `android:layout_width` et `android:layout_height` à positionner sur les composants graphiques contenus dans le `LinearLayout`
- Les valeurs possibles de ces propriétés peuvent être :
  - une valeur exacte de pixel (125px pour 125 pixels) : c'est fortement déconseillé
  - `wrap_content` qui indique que le composant prend la taille qu'il faut pour s'afficher correctement entièrement
  - `match_parent` (anciennement `fill_parent` avant l'API 8) indique que le composant remplit complètement la dimension indiquée du `LinearLayout`

Le `RelativeLayout`

- = un conteneur qui permet de placer ses contenus les uns par rapport aux autres
- C'est le conteneur proposé dans le (nouveau) outil de construction d'IHM
- Les `Views` contenues dans le `RelativeLayout` indiquent leur positionnement à l'aide de leurs attributs (dans le fichier XML de l'IHM)
- Il ne doit pas y avoir de dépendance cyclique (bon sens)
- Les `Views` indiquent leur position par rapport à la vue parente ou leurs `Views` soeurs (en utilisant leur `id`)
- Les valeurs des attributs sont soit des `boolean`, soit l'`id` d'une autre `View`

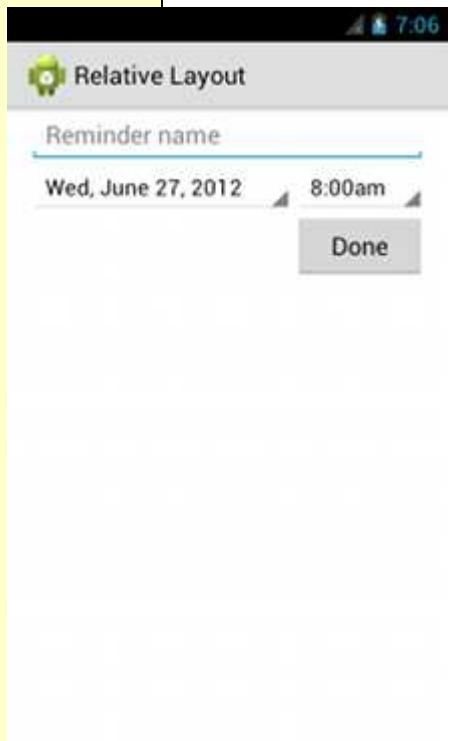
Attributs possibles pour une `View` dans un `RelativeLayout`

- Les `Views` dans un `RelativeLayout` peuvent utiliser les attributs :
  - `android:layout_alignParentTop` : si `true`, le haut de la `View` est calé sur le haut de la vue parente
  - `android:layout_centerVertical` : si `true`, la `View` est centrée verticalement à l'intérieur de la vue parente
  - `android:layout_below` : le haut de la vue est en dessous de la `View` indiquée (par son l'`id`)
  - `android:layout_toRightOf` : le coté gauche de la vue est à droite de la `View` indiquée (par son l'`id`)
- Il y a d'autres valeurs possibles voir à `http://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html`

RelativeLayout : un exemple

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```

donne :

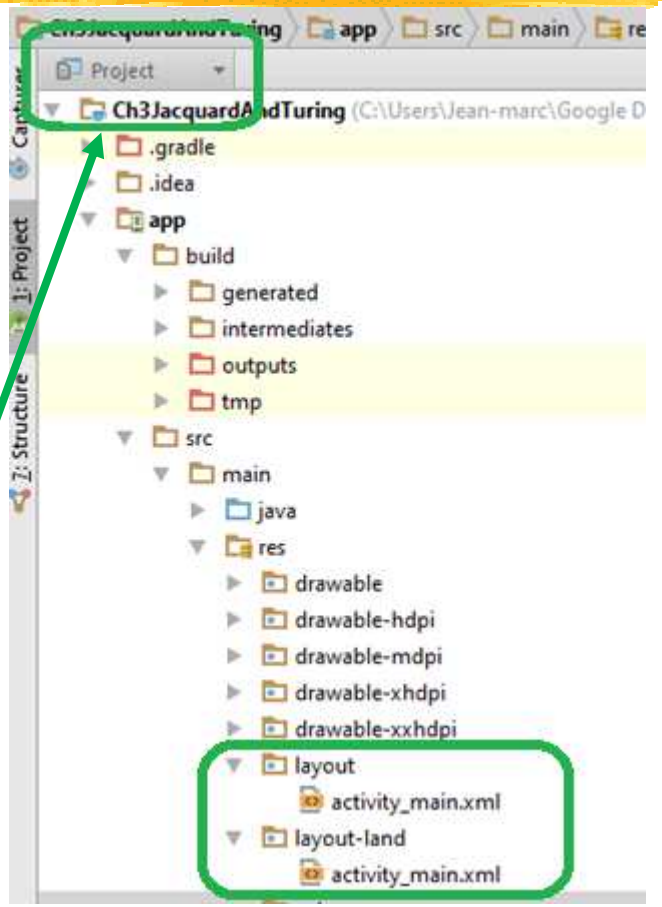


`@+id` **vs.** `@id`

- Dans le fichier xml précédent, on utilise parfois `@+id` et parfois `@id`
- `@+id` indique qu'il faut créer, si nécessaire, une nouvelle entrée dans `R.java` (et sa classe interne `id`)
- `@id` repère simplement l'identificateur `id` et il n'y a pas de création dans `R.java`
- En fait cela fonctionne si on met toujours le + !

A propos de portrait-paysage (1/2)

- Pour faire la différence entre une présentation portrait et une présentation paysage, il suffit de créer sous le répertoire `res` :
  - un répertoire `layout-land` (landscape) avec les fichiers XML d'IHM pour les présentations paysage
  - un répertoire `layout-port` (portrait) avec les fichiers XML d'IHM pour les présentations portrait
  - les fichiers par défaut sont mis dans le repertoire `layout`
- Remarque : Sélectionner l'onglet Project pour cela, pas l'onglet Android

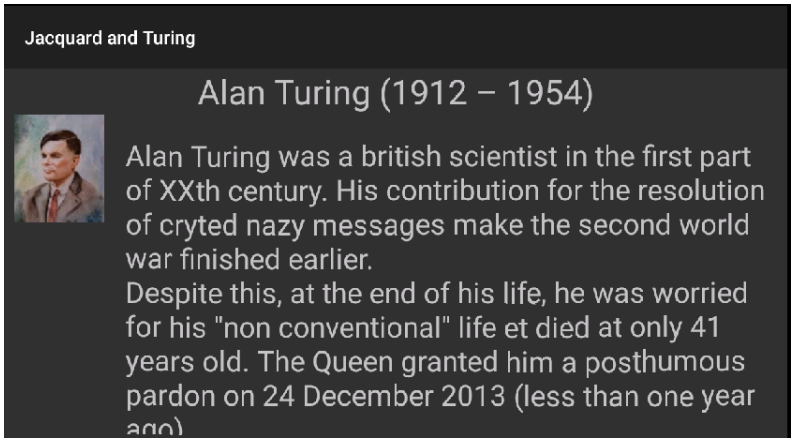


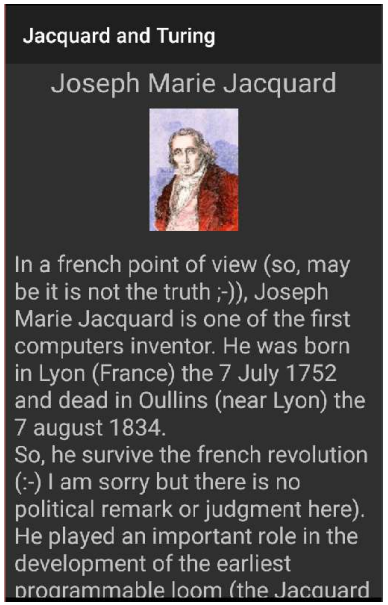
A propos de portrait-paysage (2/2)

- Ainsi, avec la seule activité :

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

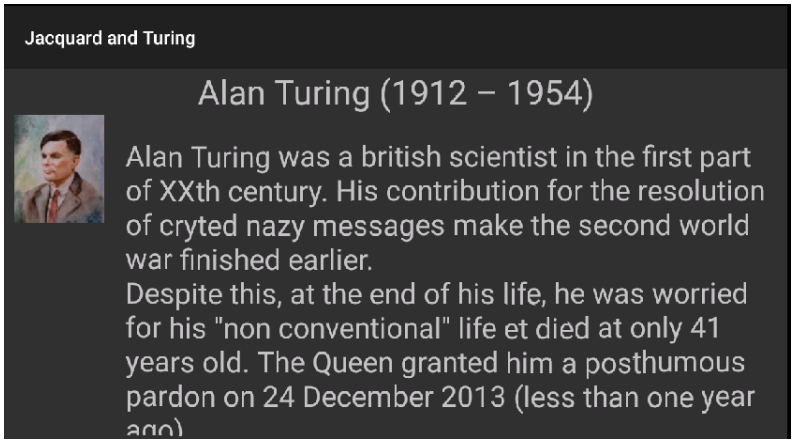
on obtient

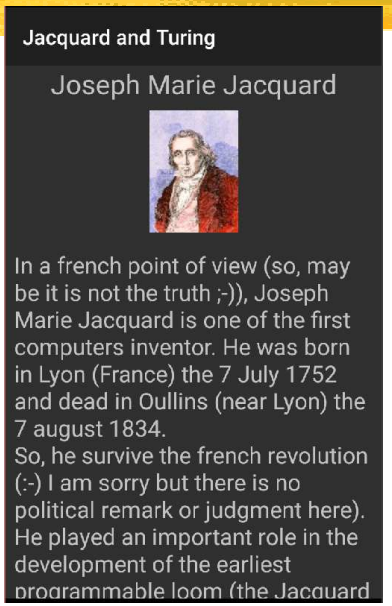




Portrait-paysage : une démo

- demo `Ch3JacquardAndTuring`





- Et si on veut un portrait-français suivi d'un paysage-anglais (pff !)
- Il faut créer des répertoires suffixés par des extensions dans un bon ordre (`layout-fr-port` et `layout-en-land`). Voir à `http://developer.android.com/guide/topics/resources/providing-resources.html`

Plus sur les ressources, `Layout`s, etc

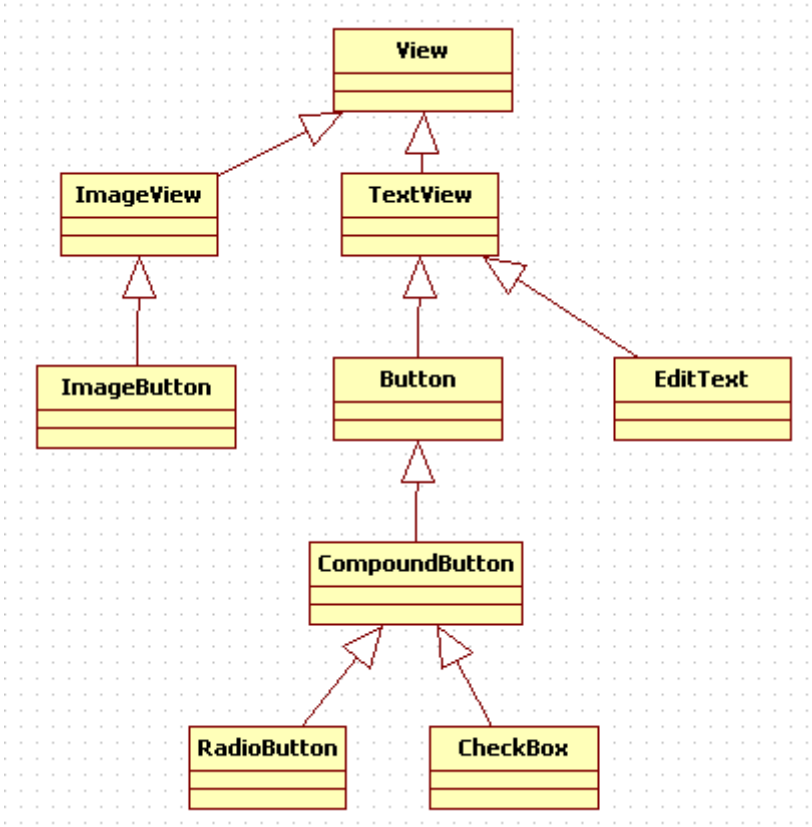
- "You can specify multiple qualifiers for a single set of resources, separated by dashes. For example, `drawable-en-rUS-land` applies to US-English devices in landscape orientation.
- The qualifiers must be in the order listed in table 2. For example:
  Wrong: `drawable-hdpi-port/`
  Correct: `drawable-port-hdpi/`"
- Bon OK, voir à
  `http://developer.android.com/guide/topics/resources/providing-resources.html`

Les contrôles Android

- Ce sont les composants graphiques que voient l'utilisateur, avec lesquels il agit sur (contrôle) l'interface graphique
- Appelés dans certains domaines, les contrôles
- En Android ce sont (par exemple) :
  - les zones de texte non éditable (~ `Label` AWT) ou éditable (~ `TextComponent` AWT) : `TextView`
  - les boutons (~ `Button` AWT) : `Button`
  - les zones de texte éditables (~ `TextField` et `TextArea` de AWT) : `EditText`
  - les cases à cocher (~ `Checkbox` AWT) : `CheckBox` et les boutons radio `RadioButton` à regrouper dans un ensemble
- Toutes ces classes sont dans le package `android.widget` et dérivent de `android.view.View`

Arborescence des principaux contrôles Android

- Les classes de composants non conteneurs (contrôles) sont rangées dans l'arbre d'héritage :



TextView

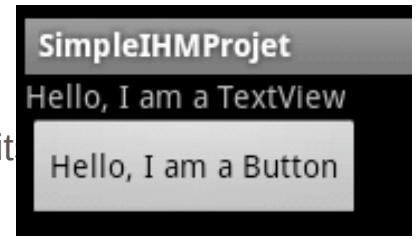
- Peut servir de zone de texte non éditable (~ `Label` AWT) et dans ce cas, sert souvent pour présenter les widgets qui suivent
- Propriétés importantes :
  - `android:text` : le texte du `TextView`
  - `android:typeface` : le type de police utilisée (monospace, ...)
  - `android:textStyle` : le style (`italic` pour l'italique, `bold_italic` pour gras et italique, ...)
  - `android:textColor` pour la couleur d'affichage du texte. Les valeurs sont en hexadécimal en unité RGB (par exemple `#FF0000` pour le rouge)

Arborescence des composants graphiques dans une IHM

- Faire une IHM c'est mettre des composants dans des composants dans des composants ... On peut le faire par programmation en utilisant `addView(View)` d'un `ViewGroup` ou dans un fichier XML
- Par exemple le fichier :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

= un `TextView` et un `Button` dans un `LinearLayout` représente l'IHM :



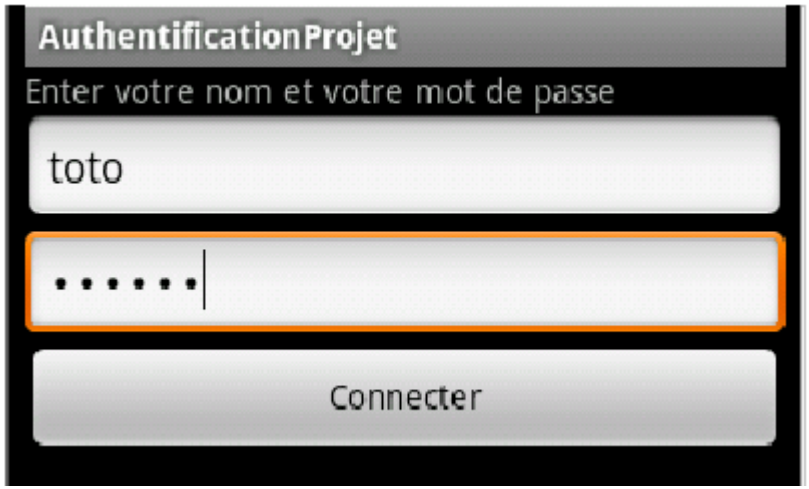
ScrollView

- Si le contenu d'une zone de texte est trop important par rapport à l'écran, on ne voit pas tout ce contenu
- L'environnement d'exécution doit pouvoir ajouter des barres de défilement à cette zone (=scrollbars). Pour cela, on met la zone de texte dans une `ScrollView`
- Le fichier XML d'IHM contient alors :

```
<ScrollView
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     ...
     android:paddingTop="8dp" >

     <TextView
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:padding="8dp"
          ...
     />
</ScrollView>
```

Exercice

- Construction d'une IHM avec des composants Android
- Construire et faire afficher l'activité :



- Remarque : le troisième (si, si !) composant n'affiche pas l'écho des caractères

La gestion des événements

- Deux moyens :
  - 1°) créer un auditeur d'événements (classe qui implémente une interface connue) et l'enregistrer auprès du composant (`View`)
  - 2°) les `View` sont elles mêmes auditrices de certains événements : (touché de l'écran). Spécialiser la méthode adaptée et lancée lorsque l'événement survient
- 1°) est classique (Java SE, Java ME). Les interfaces sont des interfaces internes à la classe `View` et de nom `OnXXXListener` (donc des interfaces de nom `View.OnXXXListener`). Cela nécessite d'implémenter une méthode de nom `onXXX()`. On enregistre un auditeur par `setOnXXXListener(View.OnXXXListener l)`
- 2°) permet d'écrire directement la gestion de certains événements qui peuvent se produire dans la `View`

Créer un auditeur d'événements : exemple

- Le code peut être :

```
private OnClickListener lAuditeurDuBouton = new OnClickListener() {
    public void onClick(View v) {
      // code lancé lorsque le bouton est cliqué
    }
};

protected void onCreate(Bundle savedValues) {
    ...
    // Récupération du Button à partir de l'IHM en XML
    Button button = (Button)findViewById(R.id.leBeauBouton);
    // Enregistrer l'auditeur auprès du bouton
    button.setOnClickListener(lAuditeurDuBouton);
    ...
}
```

Méthodes lancées par les auditeurs d'événements

- `onClick()` (de `View.OnClickListener`) est lancée lorsque l'utilisateur touche le composant graphique, ou après appui sur enter alors que le composant a le focus
- `onLongClick()` (de `View.OnLongClickListener`) : idem que si dessus mais après un appui de plus de 1 seconde
- `onKey()` (de `View.OnKeyListener`) est lancée après appui et relachement d'un touche clavier
- `onTouch()` (de `View.OnTouchListener`) est lancée pour toute action de toucher (appui, relachement, mouvement de l'utilisateur sur l'écran)
- `onCreateContextMenu()` (de `View.OnCreateContextMenuListener`) est lancée pour créer un menu contextuel

L'attribut `android:onClick`

- On peut indiquer dans le fichier `.xml` de description d'IHM, la méthode qui sera lancée sous une certaine action sur un composant graphique
- Par exemple, l'attribut `android:onClick` d'un composant graphique indique le nom de la méthode qui sera lancée si on clique sur cette `View`
- Par exemple, dans le fichier de description de l'IHM, on écrit

```
<Button
    android:id="@+id/push"
    ...
    android:onClick="onClickEmpiler">
</Button>
```

- Dans l'activité chargeant l'IHM contenant ce `Button`, on pourra écrire :

```
public void onClickEmpiler(View v){
    // traitement lancé lorsque l'utilisateur clique sur le Button d'id push
}
```

"Enchaîner" les écrans (1/2)

- Pour passer d'un écran à un autre, il faut écrire le code

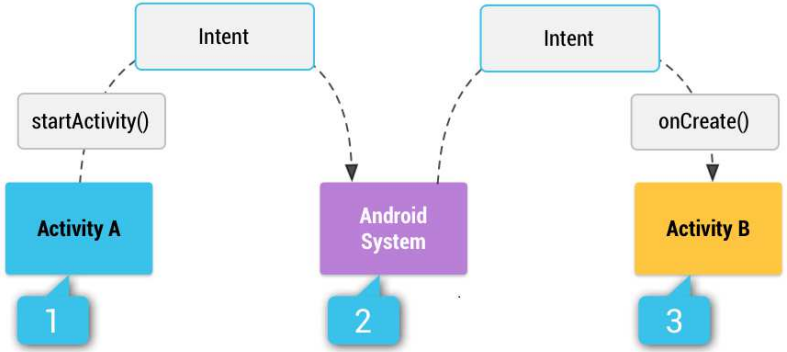
```
Intent i0 = new Intent(getApplicationContext(), NouvelleActivity.class); //1
startActivity(i0);
```

  et déclarer la nouvelle activité `NouvelleActivity.class` (le futur écran) dans `AndroidManifest.xml`

- `public void startActivity (Intent intent)` est une méthode de la classe `Activity` permettant de lancer une autre Activity (qui affichera un nouvel écran). `intent` est l'`Intent` (l'intention) qui prépare ce lancement

"Enchaîner" les écrans (2/2)

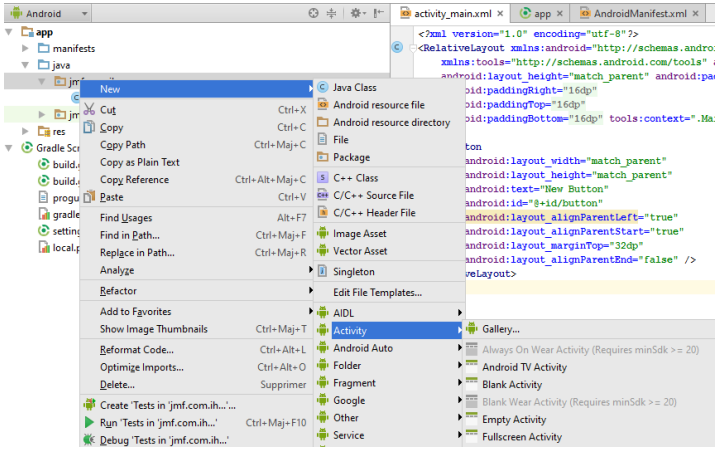
- En fait cet `Intent` est envoyé à l'environnement d'exécution. Celui-ci le redirige vers l'activité concernée



- Le premier argument du constructeur de l'`Intent` doit être le `Context`. Si l'appel est fait dans une activity `MonActivity`, `this` est souvent utilisé et convient car `Activity` dérive de `Context`
- On utilise souvent `MonActivity.this` quand on est dans un listener d'événement. Mais, la méthode `getApplicationContext()` est la plus pratique

Créer une nouvelle activité

- Dans un projet, on peut créer une nouvelle activité comme indiqué dans les diapos précédentes : écrire une classe dérivant de `Activity`, redéfinir les méthodes `onCreate()`, ... et déclarer cette activité dans l'`AndroidManifest.xml`
- Avec l'environnement de développement, si on utilise clic droit sur le répertoire java paquetage | New | Activity,



en complétant les écrans qui suivent, l'activité est en partie créée et sa déclaration correcte (fils de l'élément `application`) dans l'`AndroidManifest.xml` aussi !

```
</intent-filter>
</activity>
<activity android:name=".Main2Activity" >
</activity>
```

Passer de données entre activités grâce aux `Intent`

- Les `Intent` servent parfois d'enveloppes pour passer des informations d'une `Activity` à une autre. On utilise pour cela une des méthodes `public Intent putExtra(String nomDeLExtra, unType valeur)`
- Par exemple :

```
Intent i = new Intent(leContexte, MapStationActivity.class);
i.putExtra("latitude", latitudeDuPointCourant);
i.putExtra("longitude", longitudeDuPointCourant);
startActivity(i);
```

- Dans une `Activity`, on récupére l'`Intent` qui a lancé l'`Activity` par `getIntent()`. On peut alors récupérer tous les extras de l'`Intent` par `getExtras()`, et, par la suite, un extra associé à une entrée par `get`*`TypeEntrée`*`(nomEntrée, valeurParDefaut)`, `valeurParDefaut` est la valeur retournée si il n'y a pas d'extra associé à `nomEntrée` dans l'`Intent`
- Par exemple :

```
double laLatitudeDeLaStation =
getIntent().getExtras().getDouble("latitude", 0);
```

Un simple avertissement : `Toast`

- Une fenêtre de dialogue qui affiche un message pendant 2 (`Toast.LENGTH_SHORT`) ou 5 (`Toast.LENGTH_LONG`) secondes est un composant graphique Android : le Toast
- On le construit et on l'affiche avec le code

```
Toast leToast = Toast.makeText(leContexte, "texteAAfficher", Toast.LENGTH_LONG);
leToast.show();
```

- Une des méthodes qui construit un `Toast` est la méthode statique :
  `public static Toast makeText (Context context, CharSequence text, int duree)`
  `context` est le contexte à utiliser. En général on passe l'activité courante
  `text` est la chaîne de caractères à afficher
  `duree` est la durée d'affichage `LENGTH_SHORT` ou `LENGTH_LONG`
- Attention construire le `Toast` ne l'affiche pas : il faut utiliser `show()` pour cela
- Et donc finalement on écrit souvent :

```
Toast.makeText(leContexte, "texteAAfficher", Toast.LENGTH_LONG).show();
```

Code de "trace" en Android

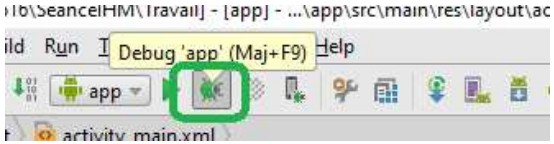
- La classe `android.util.Log` propose plusieurs méthodes de trace (de Log) hierarchisées. Ces méthodes ont pour nom une seule lettre. Ce sont, dans l'ordre les méthodes `v()` (verbose), `d()` (debug), `i()` (information), `w()` (warning) et `e()` (erreur)
- Ces méthodes ont deux arguments : (`String tag, String msg`)
- Elles permettent de visualiser des traces lors de l'exécution en utilisant l'onglet LogCat. On peut filtrer ces traces en ne laissant afficher que les log de balise `tag`
- Par exemple le code :

permet de voir les sorties dans l'onglet LogCat

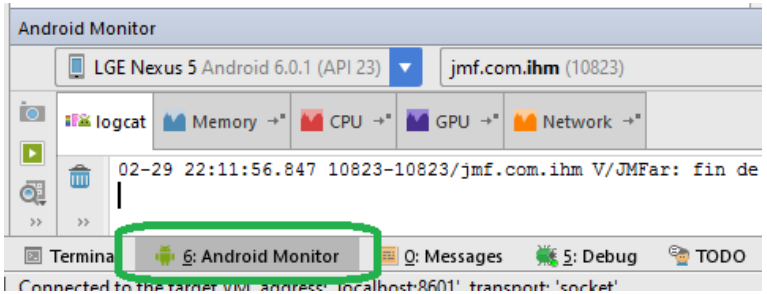
```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
     // Handle item selection
     switch (item.getItemId()) {
     case R.id.new_game:
          Log.v("JMF", "Une nouvelle partie ?");
          Log.d("JMF", "Une nouvelle partie ?");
          Log.i("JMF", "Une nouvelle partie ?");
          Log.w("JMF", "Une nouvelle partie ?");
          Log.e("JMF", "Une nouvelle partie ?");
     return true;
     // ...
     }
}
```

L'onglet LogCat

- Les traces de Log sont affichés dans l'onglet LogCat
- Pour afficher cet onglet, il faut lancer l'exécution en mode Debug : bouton



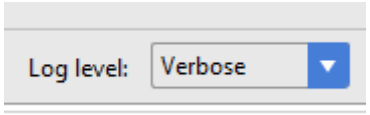
- Faire afficher l'onglet 6: Android Monitor (en bas de l'IDE)



- Il y a alors plusieurs onglets disponibles utiles lors de l'exécution de l'app dont l'onglet LogCat
- On peut faire des filtres sur les sorties dans la zone de texte de recherché de cet onglet

Traces (log) en Android

- On indique le niveau de trace dans la liste Log level :



- Le niveau verbose affichera toutes les traces du filtre, le niveau info n'affichera que les traces info, warning, erreur et assert

Exercice

- Gérer les actions des utilisateurs sur une IHM

L'internationalisation

- = i18n
- L'application doit être "localisée" = suivre la culture de l'utilisateur
- On parle de localisation
- En pratique, il faut tenir compte des différences de langues pour les textes, l'audio et les différences de présentation des nombres, des monnaies, etc.
- Une bonne solution : mettre les caractéristiques de localisation dans des fichiers de ressources plutôt que dans le code Java
- bibliographie : `http://developer.android.com/guide/topics/resources/localization.html`

La technique

- Généralement, les applications ont des fichiers de ressources par défaut. On leur ajoute des fichiers spécifiques de localisation
- A l'exécution, l'environnement choisit les fichiers adaptés à la culture (au "Locale") de l'utilisateur
- Toutes ces ressources se trouvent sous le répertoire `res` (et ses sous répertoires)

Les fichiers de ressources par défaut

- Pour les chaînes de caractères : `res/values/strings.xml`
- Pour les images : dans `res/drawable/`
- Pour les écrans d'IHM : dans `res/layout/`
- Ces fichiers doivent toujours être présents et toujours contenir toutes les ressources nécessaires à l'application
- Eventuellement on peut avoir :
  - dans `res/anim/` des animations
  - dans `res/xml/` des fichiers xml
  - dans `res/raw/` toutes sortes d'autres fichiers

Les fichiers de ressources localisés

- Ils se trouvent sous `res/`*`Repertoire`*`-`*`qualificateur`*
- *`Repertoire`* est le nom du répertoire où se trouve les ressources par défaut. Ce peut être `values`, `layout`, `drawable`, `menu`, `color`, etc.
- *`qualificateur`* est un nom spécifique de localisation. Il peut être formé de plusieurs abrévations séparées par `-`
- Plus précisément, *`qualificateur`* est défini :
  - par 2 lettres suivant le standard ISO 639-1 des codes de langues,
  - suivies éventuellement de 2 lettres de code de région suivant le standard ISO 3166-1-alpha-2 region code, précédé de la lettre `r`

Exemple de qualificateur

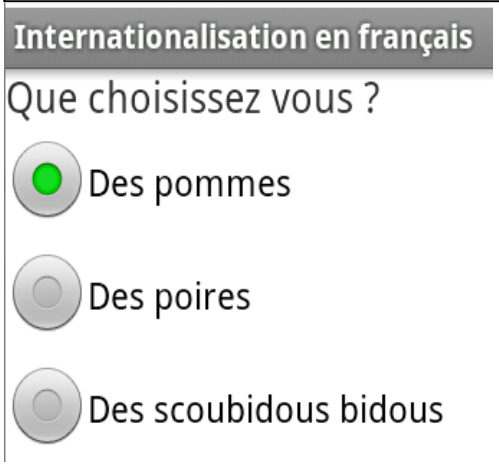
- exemple de *`qualificateur`* : `en`, `fr`, `en-rUS` (= anglais région united states = américain), `fr-rFR` (français de métropole), `fr-rCA` (français canadien), etc.
- Exemples : `res/values/strings.xml` (chaînes par défaut), `res/values-fr/strings.xml` (pour le français), `res/values-ja/strings.xml` (pour le japonais)
- bibliographie : `http://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources`

Localisation : un exemple (1/2)

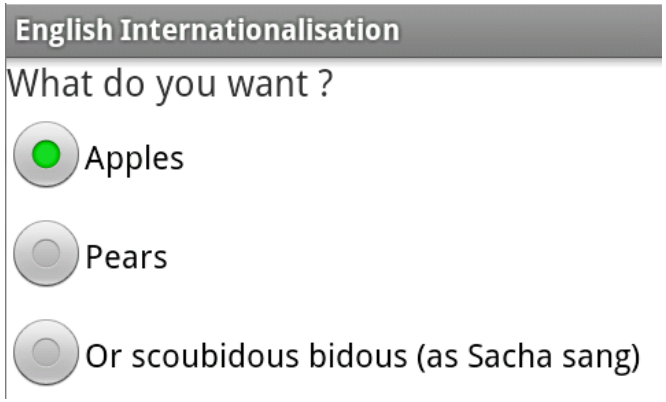
- Le "hello world" android avec le fichier `res/layout/activity_main.xml`

```
<LinearLayout ... android:orientation="vertical" ...>
    <TextView ... android:text="@string/invite" />
    <RadioGroup ...>
        <RadioButton ...
            android:text="@string/choix1" />
        <RadioButton ...
            android:text="@string/choix2" />
        <RadioButton ...
            android:text="@string/choix3" />
    </RadioGroup>
</LinearLayout>
```

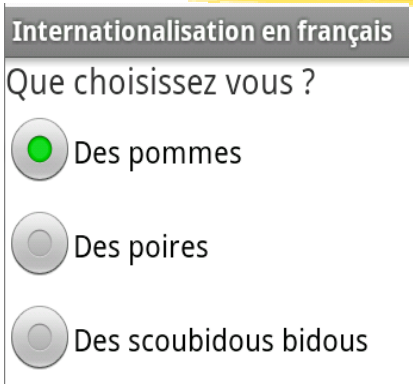
amène



ou

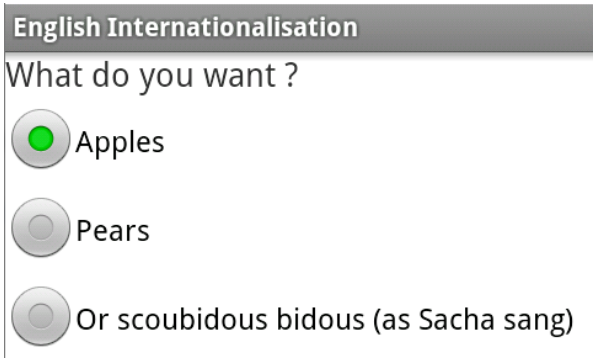


Localisation : un exemple (2/2)



utilise `res/values/strings.xml` (fichier de configuration par défaut) :

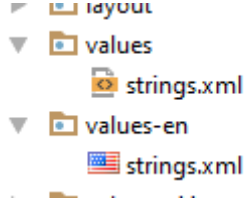
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Internationalisation en français</string>
    <string name="invite">Que choisissez vous ?</string>
    <string name="choix1">Des pommes</string>
    <string name="choix2">Des poires</string>
    <string name="choix3">Des scoubidous bidous</string>
</resources>
```



utilise `res/values-en/strings.xml` :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">English Internationalisation</string>
    <string name="invite">What do you want ?</string>
    <string name="choix1">Apples</string>
    <string name="choix2">Pears</string>
    <string name="choix3">Or scoubidous bidous (as Sacha sang)</string>
</resources>
```

- On a donc :



(utiliser l'onglet Project pas Android)

Configuration de la localisation

- Voir démo projet `Ch3InternationalisationProjet`
- Sur un AVD, choisir sur le bureau de l'AVD :
  Paramètres | Custom Locale

Trouver `fr` ou, éventuellement, l'ajouter avec le bouton "ajouter une nouvelle locale"

- Sur un Nexus S, Menu | Paramètre système | Langue et saisie, puis choisir la langue (cf. `http://www.youtube.com/watch?v=qE3B34I7tXo`)

Compléments

Les icônes

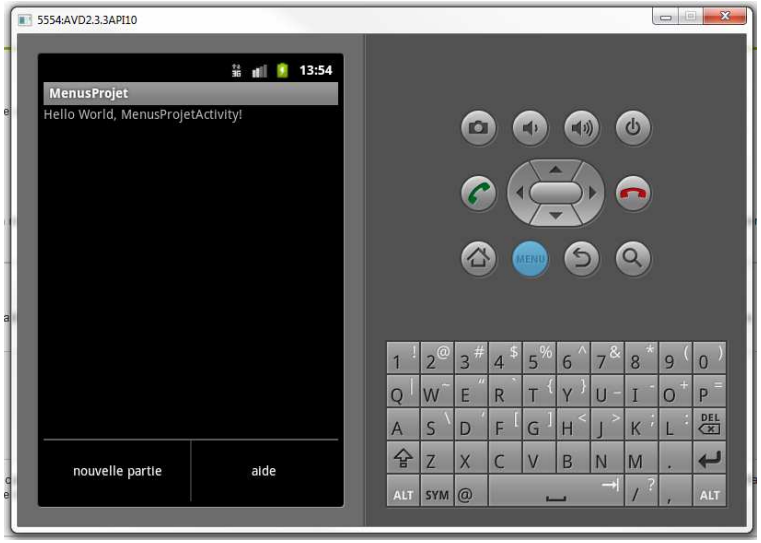
- On indique l'icône d'une application ou d'une activité principale dans l'`AndroidManifest.xml` par la balise `android:icon` :

```
<activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

- `app_icon` est le nom de base d'une fichier image sous `res/drawable`
- Pour bien faire, il est mieux de prévoir 4 fichiers images de même nom de base dans chacun des répertoires :
  `drawable-ldpi` (120 dpi, Low density screen) - icône de 36px x 36px
  `drawable-mdpi` (160 dpi, Medium density screen) - icône de 48px x 48px
  `drawable-hdpi` (240 dpi, High density screen) - icône de 72px x 72px
  `drawable-xhdpi` (320 dpi, Extra-high density screen) - icône de 96px x 96px
  pour les écrans de définition distinctes
- Bibliographie : `http://petrnohejl.github.io/Android-Cheatsheet-For-Graphic-Designers/`

Les menus

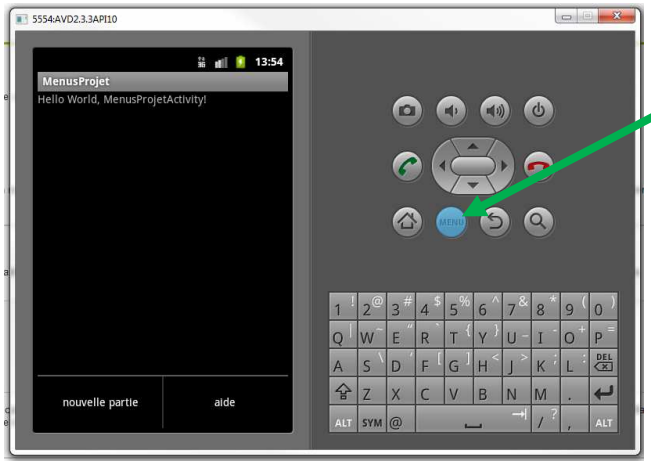
- Deux (au moins ;-)) sortes de menus :
  - menu à options = menu de base pour une activité. Il apparaît lorsque l'utilisateur presse le bouton où l'icône MENU



  - menu contextuel qui apparaît quand l'utilisateur maintient un toucher sur une `View` possédant un tel menu
- On ne crée pas un menu dans le code. On le crée dans un fichier XML. Ce fichier doit être placé dans le répertoire `res/menu`

Les menus : résultat

- Chaque activity de l'application peut avoir son menu par défaut
- Sur un AVD, le menu apparaît en cliquant sur la touche MENU



- Sur un smartphone récent (= sans bouton menu) si une activité propose un menu, l'icône de menu apparaît :

Les menus à options : le fichier XML

- biblio : `http://developer.android.com/guide/topics/ui/menus.html`
- Exemple : fichier `res/menu/mon_menu.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:title="@string/new_game" />
    <item android:id="@+id/help"
          android:title="@string/help" />
</menu>
```

- Remarque : les noms de fichiers de ressources ne doivent contenir que des lettres minuscules, des chiffres et le caractère _ (pas de majuscules)
- Evidemment il faut ajouter des valeurs associées à `new_game` et `help` dans le fichier `res/values/strings.xml` (à cause des `@string/...`)

Les menus à options : le code

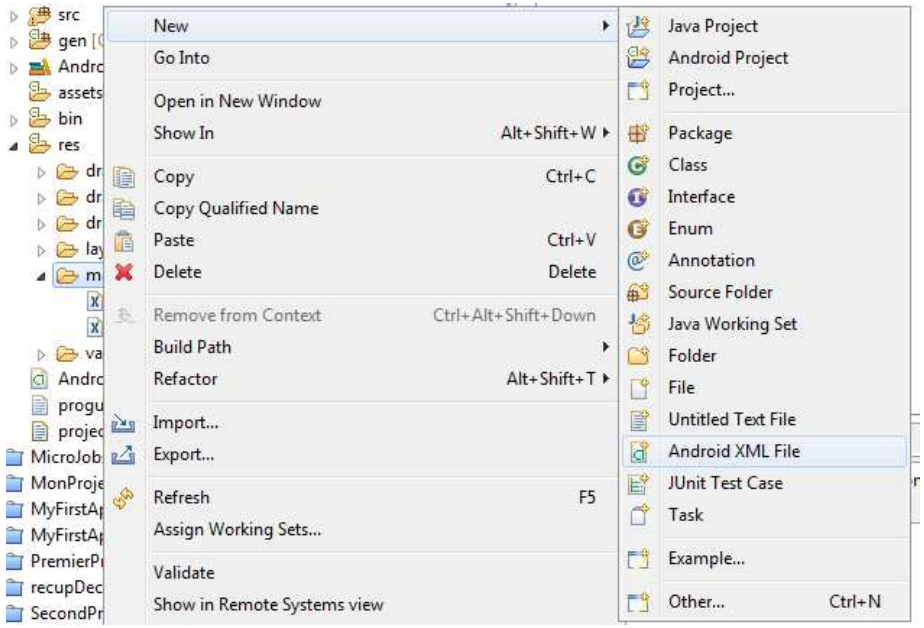
- Le code à ajouter dans l'activité est :

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
     MenuInflater inflater = getMenuInflater();
     inflater.inflate(R.menu.mon_menu, menu);
     return true;
}
```

- On récupère un `MenuInflater` qui, à l'aide de sa méthode `inflate()`, et, à partir du fichier XML `mon_menu.xml`, construit un objet de la classe `Menu` et le repère à l'aide de son second argument (ici `menu` !)
- Remarque 1 : c'est la méthode déjà donnée par l'environnement de développement
- Remarque 2 : "You must return `true` for the menu to be displayed; if you return `false` it will not be shown." !
- Remarque 3 : l'objet de la classe `Menu` étant créé, on peut lui ajouter des items à l'aide des méthodes `add()`

Les menus avec l'IDE

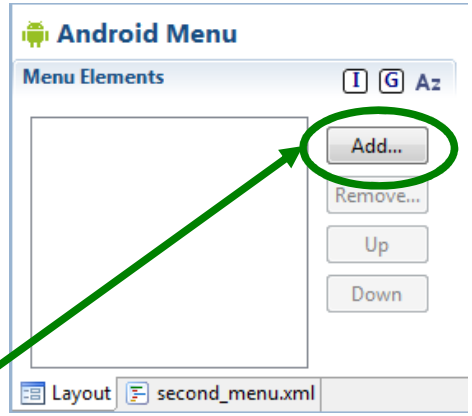
- On peut créer les menus à l'aide de l'IDE
- Sous `res/menu` cliquer droit puis New | Android XML File



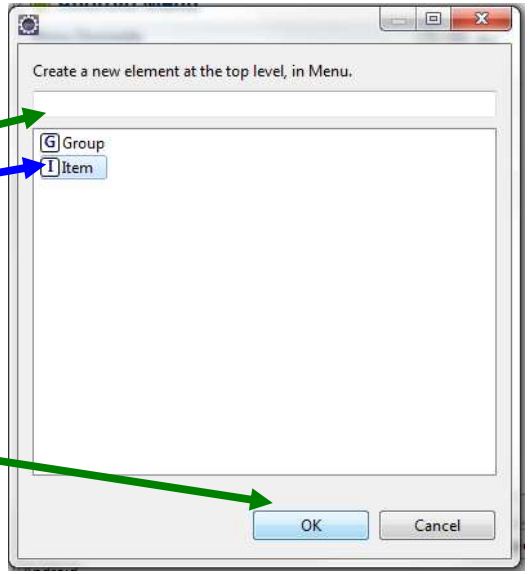
- Donner un nom de fichier menu. Cliquer Finish

Les items de menus avec l'IDE (1/2)

- On a alors une nouvelle fenêtre :

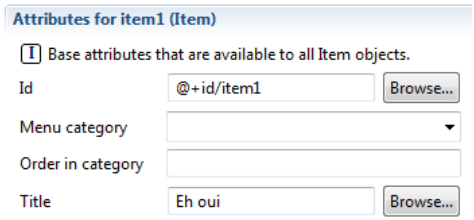


- Dans l'onglet Layout, ajouter les items du menu en cliquant Add...
- Ne pas mettre de nom dans la zone de saisie (sinon OK devient inactif)
  Sélectionner Item
  puis cliquer OK :



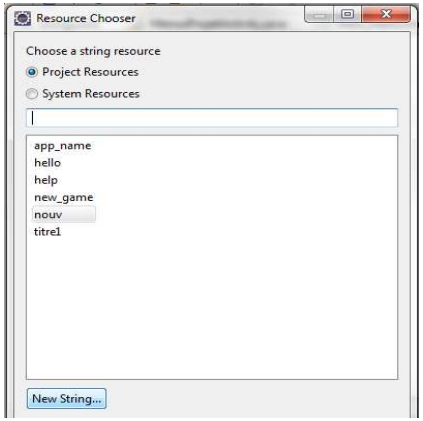
Les items de menus avec l'IDE (2/2)

- Dans le nouvel onglet Attributes for item1 (Item), indiquer un titre (dans Title)



- Cliquer à l'extérieur de l'onglet. L'item a été ajouté (le voir dans la version XML)

- Conseil pour l'internationalisation. Eviter de mettre "en dur" l'intitulé de l'item (ligne Title). Utiliser Browse qui permet de construire une nouvelle chaîne en cliquant sur New String...



- Dans la fenêtre "Create New Android String", compléter "New R.string." et "String" et une nouvelle entrée dans `res/values/strings.xml` fait la correspondance

Gestion des événements pour les menus

- La méthode qui gère les événements de sélection d'un item de menu est déjà intégrée aux activités, c'est à dire lorsqu'un item de menu est sélectionné par l'utilisateur, la méthode `public boolean onOptionsItemSelected(MenuItem item)` est automatiquement lancée. Il suffit donc de l'écrire !
- L'item sélectionné est repéré par l'argument de cette méthode
- La méthode `getItemId()` lancée sur cet argument permet de connaître l'item sélectionné en le comparant aux valeurs `android:id` des items. Lorsque le traitement est effectué correctement cette méthode doit retourner `true`
- "If you don't handle the menu item, you should call the superclass implementation of `onOptionsItemSelected()`"
- Exemple de code :


```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
     // Handle item selection
     switch (item.getItemId()) {
     case R.id.new_game:
          // un traitement;
          return true;
     case R.id.help:
          // un autre traitement
          return true;
     default:
          return super.onOptionsItemSelected(item);
     }
}
```

Les menus contextuels (1/2)

- Un menu contextuel peut être affiché lorsque l'utilisateur maintient un contact prolongé sur un composant graphique
- On indique qu'un composant a un menu contextuel associé par

```
registerForContextMenu(leComposantPossedantUnMenuContextuel);
```

  lancé dans l'activité
- Par exemple : `registerForContextMenu((TextView)findViewById(R.id.leTextView));`
- Quand le composant reçoit un appui prolongé, la méthode `public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo)` est lancée.
  `menu` est le `ContextMenu` qui va être construit,
  `v` est la vue actionnée par l'utilisateur,
  `menuInfo` sont des informations
- C'est cette méthode qui construit le menu et initialise l'objet repéré par l'argument `menu` : c'est similaire aux menus d'activité

Les menus contextuels (2/2)

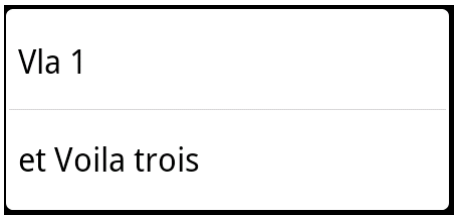
- Le code peut être :

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

- Lorsque le composant associé à ce menu contextuel est activé (clic prolongé !), le menu décrit par le fichier `res/menu/mon_menu_contextuel.xml` :

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/item1" android:title="@string/etUn"></item>
    <item android:id="@+id/item2" android:title="@string/etTrois"></item>
</menu>
```

est affiché :



- Demo dans `Ch3MenusProjet`

Gestion des événements pour les menus contextuels

- Lorsqu'un item de menu contextuel est sélectionné par l'utilisateur, l'activité lance automatiquement la méthode `public boolean onContextItemSelected(MenuItem item)`
- L'item sélectionné est repéré par l'argument de cette méthode
- La méthode `getItemId()` lancée sur cet argument permet de connaître l'item sélectionné en le comparant aux valeurs `android:id` des items. Lorsque le traitement est effectué correctement cette méthode doit retourner `true`
- Bref comme pour les items de menus à options

- Demo toujours dans `Ch3MenusProjet!`

Sous menus

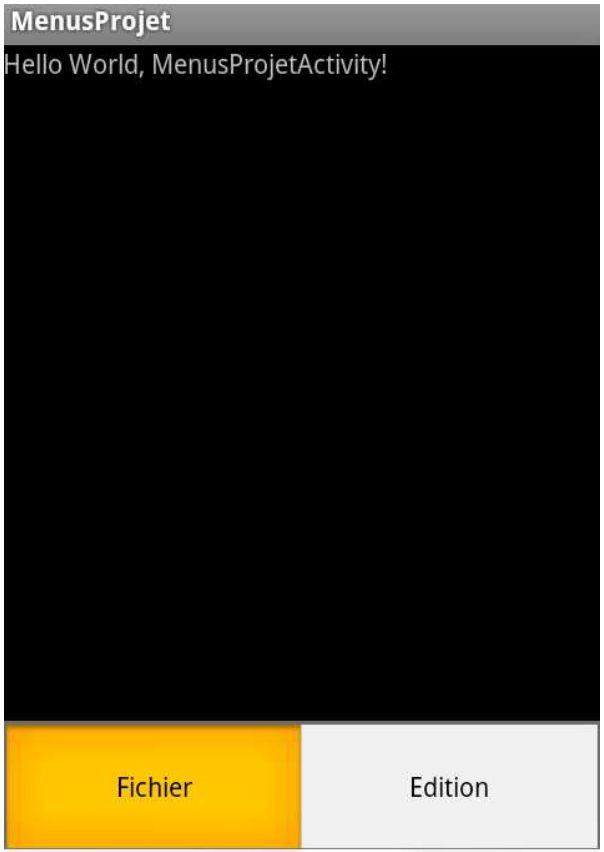
- On crée un sous menu à partir d'un item de menu à l'aide de la balise menu comme sous balise de la balise item :

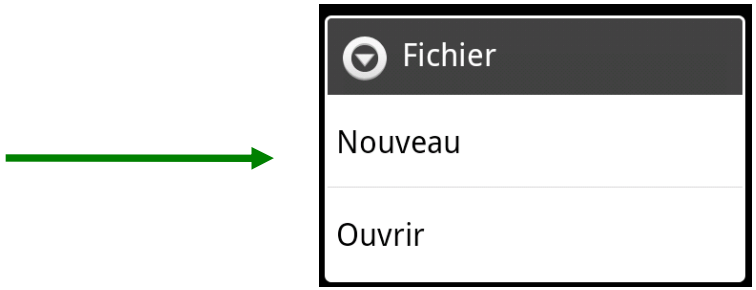
```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/item1"
        android:title="@string/menufichier">
        <menu>
            <item
                android:id="@+id/create_new"
                android:title="@string/create_new"/>
            <item
                android:id="@+id/open"
                android:title="@string/open"/>
        </menu>
    </item>
    <item
        android:id="@+id/item2"
        android:title="@string/menuedition">
        <menu>
            <item
                android:id="@+id/cut"
                android:title="@string/cut"/>
            <item
                android:id="@+id/copy"
                android:title="@string/copy"/>
        </menu>
    </item>
</menu>
```

- Il suffit alors de charger ce fichier dans l'activité dans la méthode `onCreateOptionsMenu()`

Sous menus : exécution

- On obtient :





- Démo : encore et toujours `Ch3MenusProjet`

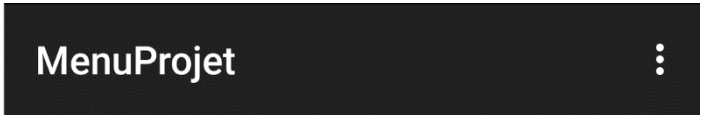
Les menus à options depuis android 3.0

- Les smartphones récents (>= Android 3.0) n'ont plus de bouton menu
- Depuis Android 3.0, les items d'un menu à options peuvent être accessibles à partir de l'action bar de l'activité courante à partir de l'icône :

- Les codes précédents des menus à options fonctionnent encore et affiche cette action bar !
- Par défaut, tous les items apparaissent en cliquant cette icône
- Si on veut faire apparaître certains items directement dans l'action bar, il faut ajouter `android:showAsAction="ifRoom"` comme attribut de la balise `item` de cet item. `ifRoom` = si possible
- Démo : `Ch3ActionBar`

L'ActionBar

- Elle est parfois (mal) nommée app bar. Mais elle est propre à chaque activité
- Elle peut être plus riche que simplement le titre de l'activité et le bouton "overflow menu" :



- biblio : `http://developer.android.com/training/appbar/index.html`
- En général, une action bar a une icône, un titre, des actions principales, d'autres actions

L'ActionBar : gestion des diverses versions

- Pour gérer les diverses versions Android, on peut mettre dans le fichier `styles.xml` du répertoire `values` le thème :

```
<resources>
     <style name="AppTheme" parent="android:Theme.Light" />
</resources>
```

  Ce sera le fichier de styles par défaut

- Dans les répertoires `values-v11` et `values-v14`, on a deux fichiers `styles.xml` de même contenu :

```
<resources>
    <style name="AppTheme" parent="android:Theme.Holo.Light.DarkActionBar" />
</resources>
```

  permettant d'avoir une `ActionBar` plus adaptée

- biblio : `http://www.tutos-android.com/actionbar-menu-android`

ActionBar : un exemple (1/2)

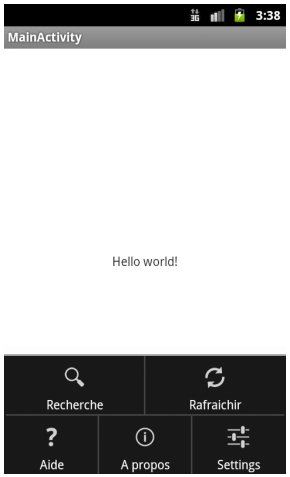
- Avec un fichier de menu (dans `res/menu`) contenant :

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
<item
     android:id="@+id/menu_search"
     android:icon="@drawable/ic_action_search"
     android:showAsAction="ifRoom"
     android:title="@string/menu_search"/>
<item
     android:id="@+id/menu_refresh"
     android:icon="@drawable/ic_action_refresh"
     android:showAsAction="ifRoom"
     android:title="@string/menu_refresh"/>
<item
     android:id="@+id/menu_help"
     android:icon="@drawable/ic_action_help"
     android:showAsAction="ifRoom"
     android:title="@string/menu_help"/>
<item
     android:id="@+id/menu_about"
     android:icon="@drawable/ic_action_about"
     android:showAsAction="never"
     android:title="@string/menu_about"/>
<item
     android:id="@+id/menu_settings"
     android:icon="@drawable/ic_action_settings"
     android:showAsAction="never"
     android:title="@string/menu_settings"/>
</menu>
```

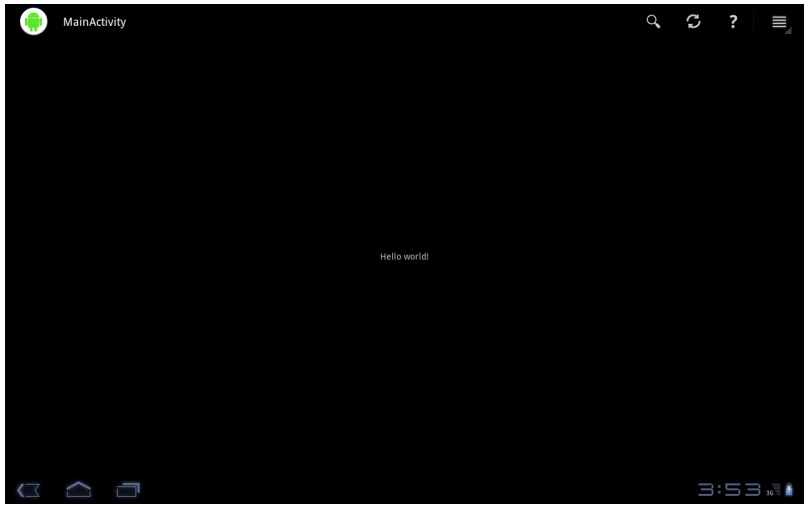
ActionBar : un exemple (2/2)

- On obtient avec :

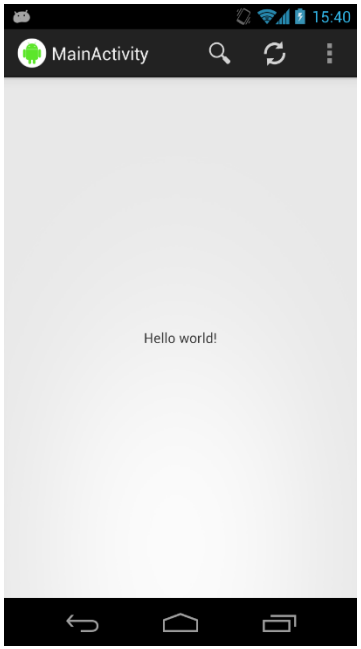
Android 2.3,



Android 3.0,



Android 4.1



- Remarques : voir les items affichés fonctions de la valeur de `android:showAsAction` (`ifRoom`, `never`, ...)

Les menus à options : la suite

- Depuis la version 5.0 d'Android (Lollipop), on peut utiliser les Toolbar à la place des ActionBar !
- Cette Toolbar est présentée dans le fichier layout de l'activité => c'est mieux non ? Par exemple :

```
<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=".MyActionBarActivity">

 <android.support.v7.widget.Toolbar xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:background="#58FA58"
        android:minHeight="?android:attr/actionBarSize" />
...
```

- Elle est mise dans l'activité par :

```
Toolbar toolbar =
     (Toolbar) findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
```

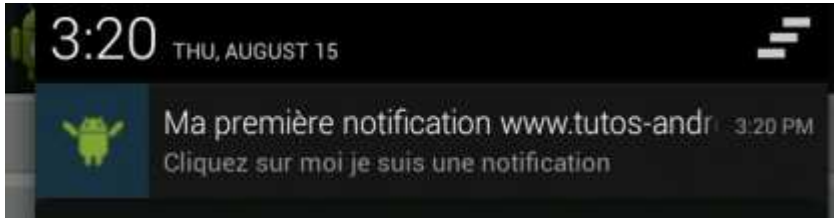
- biblio :

http://www.tutos-android.com/remplacer-actionbar-toolbar-android,
http://www.vogella.com/tutorials/AndroidActionBar/article.html

Notification

- Une notification est une indication placée dans la barre des notifications :

- Pour voir le détail de la notification, déplacer cette barre vers le bas :



- Evidemment ces IHM dépendent de la version Android utilisée
- Les applications et l'environnement d'exécution peuvent utiliser les notifications pour informer l'utilisateur (arrivée de SMS, …)
- Voir démo projet `Ch3NotificationProjet`

Création d'une notification

```
private final void createNotification(){
     //Création de la notification avec spécification de l'icône de la notification et
     //le texte qui apparait à la création de la notification
     //Récupération du titre et description de la notification
     String notificationTitle = getResources().getString(R.string.notification_title);
     String notificationDesc = getResources().getString(R.string.notification_desc);

     NotificationCompat.Builder mBuilder =
          new NotificationCompat.Builder(this)
               .setSmallIcon(R.drawable.notification)
               .setContentTitle(notificationTitle)
               .setContentText(notificationDesc);

     // Définition de la redirection au moment de la sélection de la notification.
     // Dans notre cas la notification redirige vers notre application
     PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, new Intent(this,
               TutoNotificationHomeActivity.class), 0);

     // Construction de la notification et de ses caractéristiques (vibration, ...)
     mBuilder.setContentIntent(pendingIntent);
     Notification notification = mBuilder.build();

     notification.vibrate = new long[] {0,200,100,200,100,200};

     //Récupération du notification Manager
     NotificationManager notificationManager =
          (NotificationManager)getSystemService(Context.NOTIFICATION_SERVICE);
     notificationManager.notify(NOTIFICATION_ID, notification);
}
```

Construction d'une notification

- On construit une notification à l'aide d'un `NotificationCompat.Builder`
- Euh au début on utilisait le constructeur `Notification(...)` puis on a utilisé `Notification.Builder(...)` !
- Lorsque l'utilisateur sélectionne la notification, il sera redirigé vers une activité décrite par un `PendingIntent` qui est construit par la méthode
  `public static PendingIntent getActivity (Context context, int requestCode, Intent intent, int flags)`
  `requestCode` n'est, en fait, pas utilisé
  `intent` est créé par `new Intent(...)` dont le second argument est l'activité qui sera lancée
  `flags` indique comment doit être utilisé l'intent envoyé à l'activité. `0` convient très bien !
- Le `NotificationCompat.Builder` permet de construire la notification (avec `build()`) et de l'assembler avec le `PendingIntent` (avec `setContentIntent()`)

Faire vibrer le smartphone et avertir le `NotificationManager`

- Par la suite, on indique que la notification devra faire vibrer le smartphone par :
```
notification.vibrate = new long[] {0,200,100,200,100,200};
```
- Pour cela, on utilise la donnée membre publique (beurk) `vibrate` de la notification. On l'initialise par un tableau de `long`, indiquant les durées de vibration et non-vibration en millisecondes
- Il faut aussi ajouter
```
<uses-permission
android:name="android.permission.VIBRATE" />
```
dans le fichier `AndroidManifest.xml`
- Enfin on indique au `NotificationManager` (qu'on a récupéré par `getSystemService(Context.NOTIFICATION_SERVICE)`) qu'il gère la notication d'ID `NOTIFICATION_ID`

Détruire une notification

```
private void deleteNotification(){
     NotificationManager notificationManager =
          (NotificationManager)getSystemService(Context.NOTIFICATION_SERVICE);
     //la suppression de la notification se fait grâce à son ID
     notificationManager.cancel(NOTIFICATION_ID);
}
```

- Il suffit de lancer la méthode `cancel()` en passant l'ID de la notification au `NotificationManager`
- bibliographie : `http://nbenbourahla.developpez.com/tutoriels/java/android_notification/`
- Compléments sur les notifications :
  Voir à `http://developer.android.com/training/notify-user/index.html`, ainsi que `http://developer.android.com/guide/topics/ui/notifiers/notifications.html`

Notification : bibliographie

- Bibliographie sur les notifications :
- `http://nbenbourahla.developpez.com/tutoriels/java/android_notification/`,
- et aussi : `http://developer.android.com/training/notify-user/index.html`, `http://developer.android.com/guide/topics/ui/notifiers/notifications.html`
- Ce sont des composants utilisables à partir d'Android 3.0 (API 11)

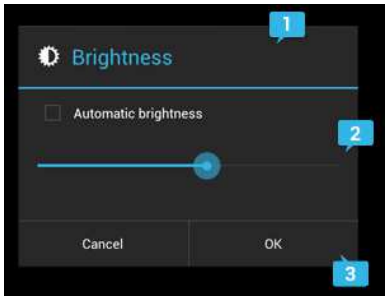
Boîtes de dialogues

- "A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed." (*)
- C'est donc plutôt modal
- "Dialogs prompt the user for decisions or additional information required by the app to continue a task. Such requests can range from simple Cancel/OK decisions to more complex [requests]" (**)
- Les alertes, les popups, les toasts sont aussi considérés comme des boîtes de dialogues
- (*) : `http://developer.android.com/guide/topics/ui/dialogs.html` (pour la programmation des boîtes de dialogues)
- (**) : `http://developer.android.com/design/building-blocks/dialogs.html` (pour l'utilisation des boîtes de dialogues)

Exemple de boîtes de dialogues

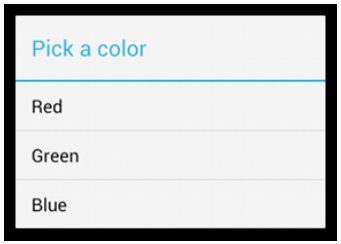
- Ce sont des composants utilisables à partir d'Android 3.0 (API 11)
- Il y a les dialogues simples :

les alertes :

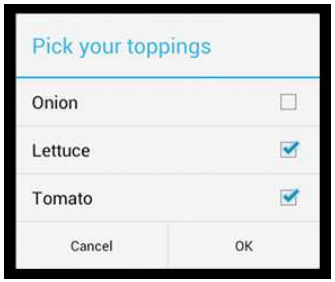


constituées du titre (1), du corps (2), des boutons d'actions (3)

les listes : simple choix



, à choix persistents multiples ou uniques



- On peut aussi créer ses propres dialogues :

Construction d'une boîte de dialogue : le code

- Démo dans Ch3DialogProjet
- On crée 

, grâce à la classe :

```
public class FireMissilesDialogFragment extends DialogFragment {
     @Override
     public Dialog onCreateDialog(Bundle savedInstanceState) {
          AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
          builder.setMessage(R.string.dialog_fire_missiles)
               .setPositiveButton(R.string.fire, new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                         Toast leToast = Toast.makeText(getApplicationContext(),
                              "Feu à volonté", Toast.LENGTH_LONG);
                         leToast.show();
                    }
               })
               .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                         Toast leToast = Toast.makeText(getApplicationContext(),
                              "Ben, j'hésite encore", Toast.LENGTH_LONG);
                         leToast.show();
               }
          });
          return builder.create();
     }
}
```

Le code de construction : explication

- La classe permettant d'obtenit le dialogue doit dériver de `android.support.v4.app.DialogFragment`
- Dans `public Dialog onCreateDialog(Bundle savedInstanceState)`, on utilise `AlertDialog.Builder` qui retourne un builder
- Les méthodes `setXXX()` de cette classe retourne l'instance d'où l'enchaînement des appels

Affichage d'une boîte de dialogue : le code

- Pour faire afficher la boite de dialogue, la méthode `afficheFireDialog()` ci dessous suffit :

```
public class MainActivity extends FragmentActivity {
     ...
     private void afficheFireDialog(){
          DialogFragment newFragment = new FireMissilesDialogFragment();
          FragmentManager leFragmentManager = getSupportFragmentManager();
          newFragment.show(leFragmentManager, "missiles");
     }
}
```

- Mais pour avoir la méthode `getSupportFragmentManager()`, qui retourne un `android.support.v4.app.FragmentManager`, l'activity doit être une `android.support.v4.app.FragmentActivity`
- Le second argument de `show()` est une chaîne de caractères nécessaire au `FragmentManager`

Une boîte de dialogue personnalisée (1/2)

- On veut créer la boite de dialogue :

- Son IHM est (évidemment) créé dans un fichier XML : `res\layout\authentification_dialog.xml`

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

    <EditText
        android:id="@+id/username"
        android:inputType="textEmailAddress"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
     ...
        android:hint="@string/username" />
    <EditText
        android:id="@+id/password"
        android:inputType="textPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
     ...
        android:fontFamily="sans-serif"
        android:hint="@string/password"/>
</LinearLayout>
```

Une boîte de dialogue personnalisée (2/2)

- Comme précédemment, la classe qui crée la boite de dialogue :
  - dérive de `android.support.v4.app.DialogFragment`
  - implémente `public Dialog onCreateDialog(Bundle b)` est obtenu par le code :

```
public class AuthentificationDialog extends DialogFragment {
...
     public Dialog onCreateDialog(Bundle savedInstanceState) {
          AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

          LayoutInflater inflater = getActivity().getLayoutInflater();

          View v = inflater.inflate(R.layout.authentification_dialog, null);
          final EditText name_place = (EditText)v.findViewById(R.id.username);
          final EditText description_place = (EditText)v.findViewById(R.id.password);
          builder.setView(v)
          .setTitle(R.string.shareTitle)
          ... // suite du code diapo suivante
          return builder.create();
     }
}
```

Réaction à l'utilisation des boutons

- La réaction à l'appui sur les boutons OK et Annuler de :

a été programmé dans le builder en lançant :

```
.setPositiveButton(R.string.ok, new DialogInterface.OnClickListener() {
     @Override
     public void onClick(DialogInterface dialog, int id) {
          String name = name_place.getText().toString();
          String description = description_place.getText().toString();
          mListener.onDialogPositiveClick(AuthentificationDialog.this, name, description);
     }
})
.setNegativeButton(R.string.annuler, new DialogInterface.OnClickListener() {
     public void onClick(DialogInterface dialog, int id) {
          mListener.onDialogNegativeClick(AuthentificationDialog.this);
          AuthentificationDialog.this.getDialog().cancel();
     }
});
```

sur le builder

Récupérer des données d'une boîte de dialogue (1/2)

- Il suffit donc que l'activité qui a ouvert cette boite soit un listener de la boite de dialogue
- Pour cela, a été défini dans la classe dialogue l'interface :

```
public interface AuthentificationDialogListener {
     public void onDialogPositiveClick(DialogFragment dialog, String name, String description);
     public void onDialogNegativeClick(DialogFragment dialog);
}
```

- Lorsque la boite de dialogue est construite sa méthode `onAttach(...)` est lancée. On indique que l'activité est un listener de la boite à cet endroit :

```
AuthentificationDialogListener mListener;

@Override
public void onAttach(Activity activity) {
     super.onAttach(activity);
     try {
          mListener = (AuthentificationDialogListener) activity;
     } catch (ClassCastException e) {
          throw new ClassCastException(activity.toString()
               + " must implement ShareDialogListener");
     }
}
```

- Remarque : on peut toujours repéré un objet d'une classe quelconque par une référence d'une interface quelconque (eh oui jean-marc !)

Récupérer des données d'une boîte de dialogue (2/2)

- Quant à l'activité, elle a été codée comme :

```
public class MainActivity extends FragmentActivity implements
     AuthentificationDialog.AuthentificationDialogListener {
     public void onDialogPositiveClick(DialogFragment dialog, String name, String description) {
          Toast.makeText(this, "Position Shared: " + name + ", " +
               description,Toast.LENGTH_SHORT).show();
     }

     public void onDialogNegativeClick(DialogFragment dialog) {
          Toast.makeText(this, "Cancel Share", Toast.LENGTH_SHORT).show();
     }

     private void afficheAuthentificationDialog(){
          DialogFragment newFragment = new AuthentificationDialog();
          FragmentManager leFragmentManager = getSupportFragmentManager();
          newFragment.show(leFragmentManager, "Authentification");
     }
     ...
}
```

et récupère bien le login et mot de passe

- Remarque : évidemment l'activité implémente l'interface `AuthentificationDialog.AuthentificationDialogListener` (cf. remarque diapo précédente) et est donc un listener qui va lancer les méthodes `onDialogXXXClick()` au moment opportun (au click sur les boutons du dialogue)

Multi-plateformes

- Internationalisation, diverses définitions d'écran (cf. icônes), voir par ailleurs
- Il faut traiter les versions diverses Android des smartphones
- Les diverses versions d'Android utilisées sont affichées dans le "dashboard" à `http://developer.android.com/about/dashboards/index.html`
- Android est "plutôt" compatible ascendant. Pour les ajouts voir à `http://developer.android.com/tools/support-library/index.html`
- Rappel : dans l'`Androidmanifest.xml` d'une application, est indiqué la version minimale d'utilisation de l'application et la version sur laquelle l'application a été testée :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" ... >
    <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="15" />
    ...
</manifest>
```

Tester la version de l'API à l'exécution

- On utilise les constantes de la classe `android.os.Build.VERSION_CODES`. Par exemple :

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        // code utilisable à partir de HONEYCOMB (Android 3.2, API 13)
}
```

- Bibliographie : `http://developer.android.com/training/basics/supporting-devices/platforms.html`

Résumé du chapitre 3 (1/2)

- Les interfaces humain machine (IHM) sont plus riches sur les smartphones que sur les ordinateurs. Elles nécessitent des API particulière : c'est le cas pour Android
- La programmation des IHM suit deux principes, l'un sur les composants graphiques, l'autre sur les classes
- On peut construire, en Android, des IHM par programmation (rarement) ou par description à l'aide fichiers XML (très souvent)
- Les conteneurs Android sont des `Layout`
- Il faut prévoir les actions de l'utilisateur sur une IHM : c'est la gestion des événements
- Dans l'utilisation d'une application Android, les écrans s'enchaînent les uns à la suite des autres

Résumé du chapitre 3 (2/2)

- On peut utiliser les `Toast` et les traces (`Log`) pour l'aide au développement
- L'internationalisation est la particularité d'adapter les applications à la culture de l'utilisateur
- Une application est souvent accessible à l'aide des icônes. Elles ont souvent des menus, des boîtes de dialogues, et peuvent laisser des notifications
- On peut connaître la plateforme d'exécution de l'application Android

Fin