



Chapitre

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 styles et les thèmes

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
- ☐ 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
- ❑ Utilise les `Intent` pour lancer d'autres activités

Construction d'une IHM

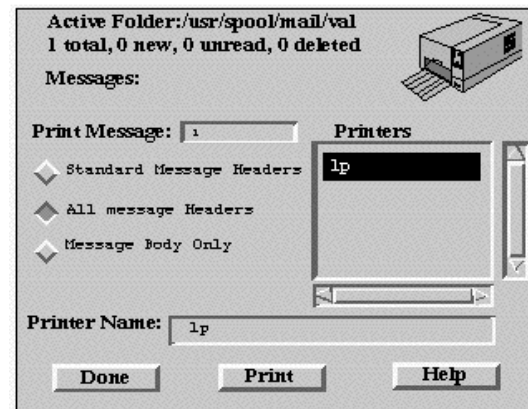


- ☐ Plutôt en XML mais
- ☐ XML ne peut pas être débogué !
- ☐ Tout ne peut pas être fait en XML

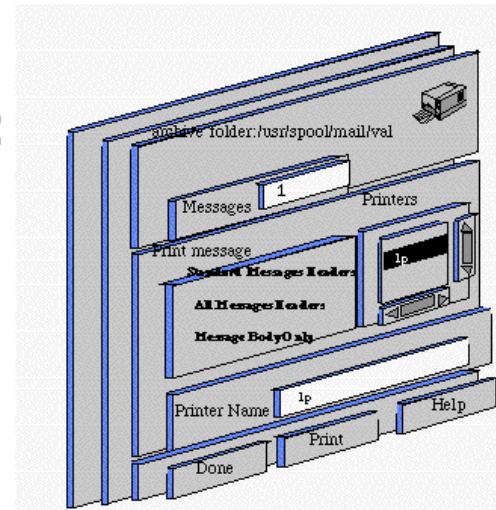
Premier principe des IHM

(1/4)

□ Quand on voit ceci :



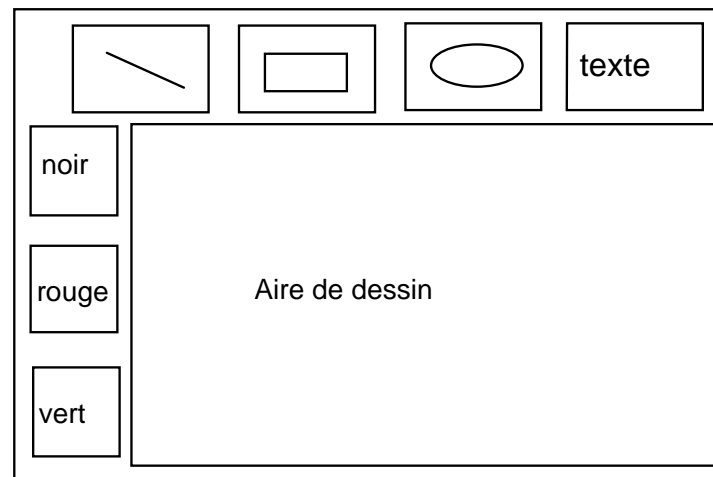
□ C'est qu'on a programmé cela :



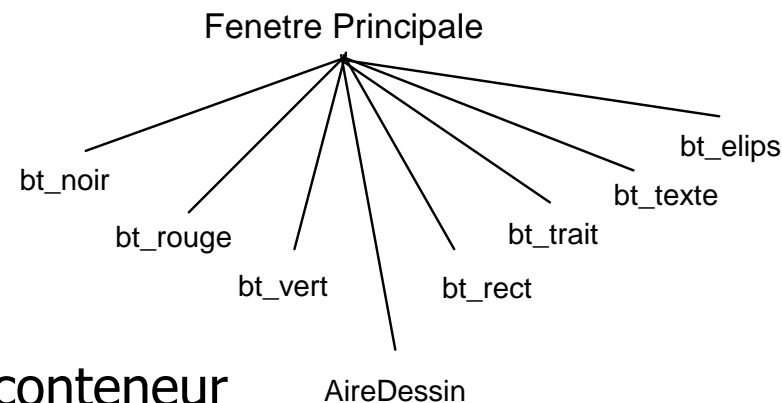
Premier principe des IHM

(2/4)

□ Si on veut :



□ On doit construire cela :

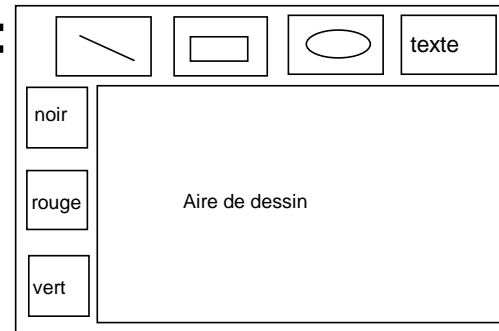


□ => Fenetre principale = un conteneur

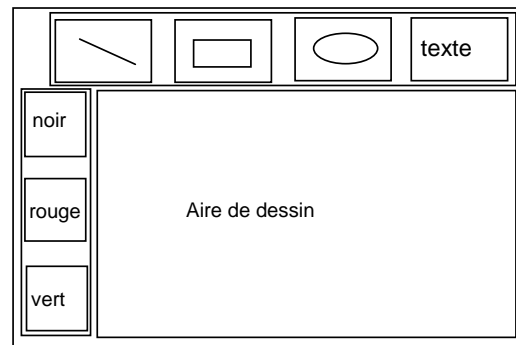
Premier principe des IHM

(3/4)

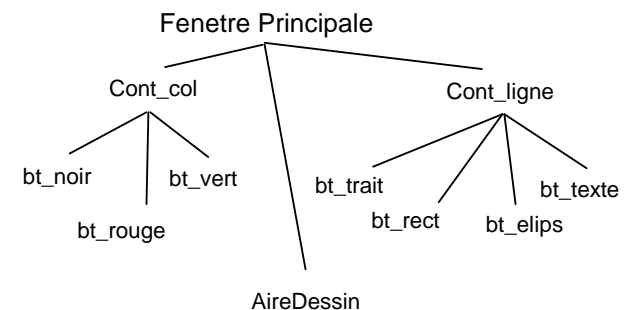
❑ Plus sûrement, si on veut :



on écrit plutôt :



c'est à dire :



Premier principe des IHM

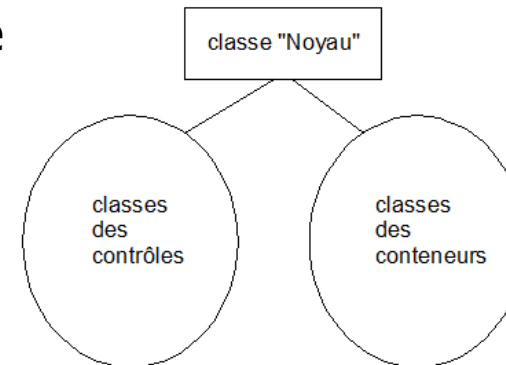
(4/4)

- ❑ (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 :

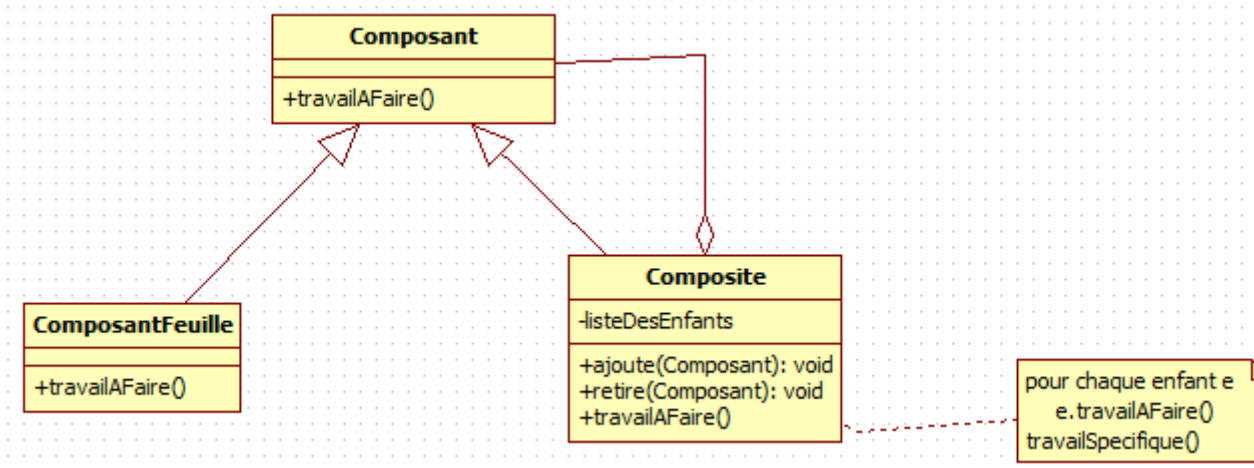


- ❑ Question : Comment sont rangées ces classes ?
- ❑ Réponse : dans un arbre d'héritage de classes : merci l'OO (bis) !

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

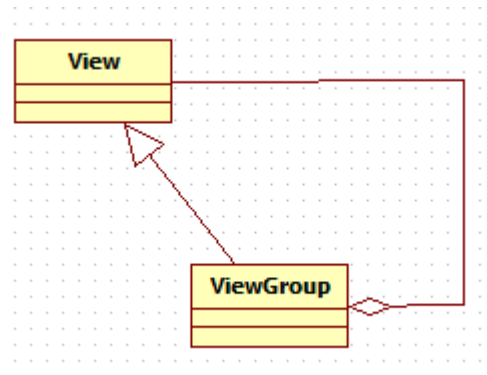
- ❑ Un composant graphique a 3 parties :
 - ❑ les données qu'il représente : le modèle (model)
 - ❑ le dessin d'affichage : la vue (view)
 - ❑ ce qui prend en charge les actions de l'utilisateur sur ce composant : le controleur (controler)
- ❑ C'est l'architecture MVC
- ❑ "L'idée est de bien séparer les données, la présentation et les traitements" (*)

❑ (*) 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)
- ❑ La seconde est courante dans certains domaines (Apple, ...)

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 icle) {
        super.onCreate(icle);
        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
```

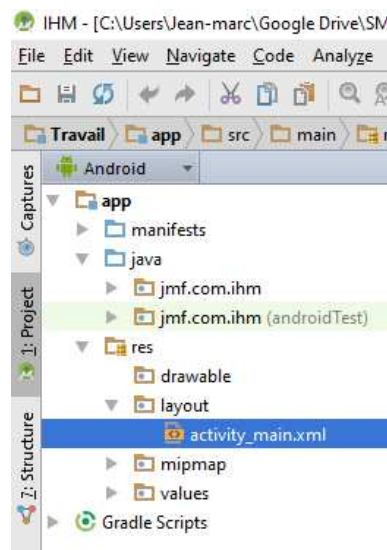
- ❑ Ce qui nécessite d'implémenter la méthode :

```
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)`)

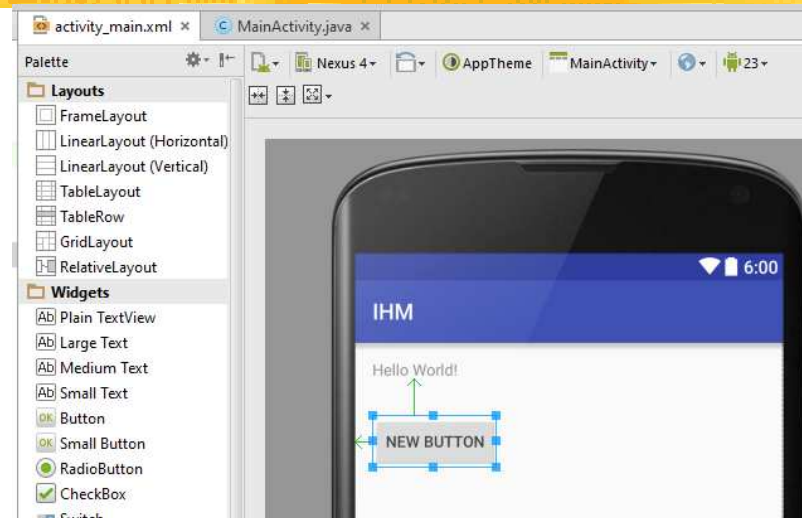
Second programme : une autre solution (1/5)

- ❑ 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 !



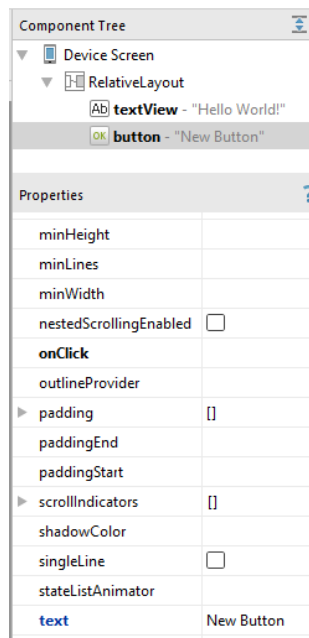
Second programme : une autre solution (2/5)

- ❑ 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



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`

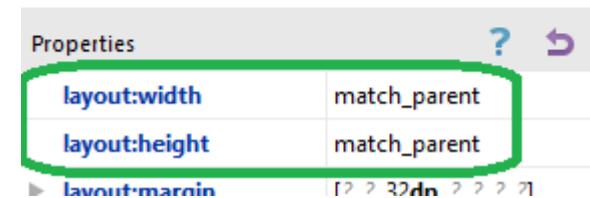
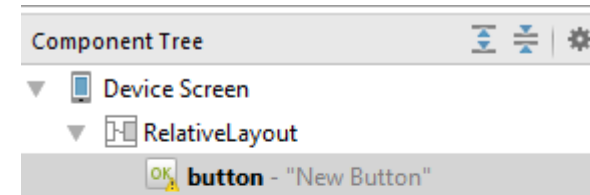
- ❑ Les valeurs de ces propriétés peuvent être

- ❑ `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>



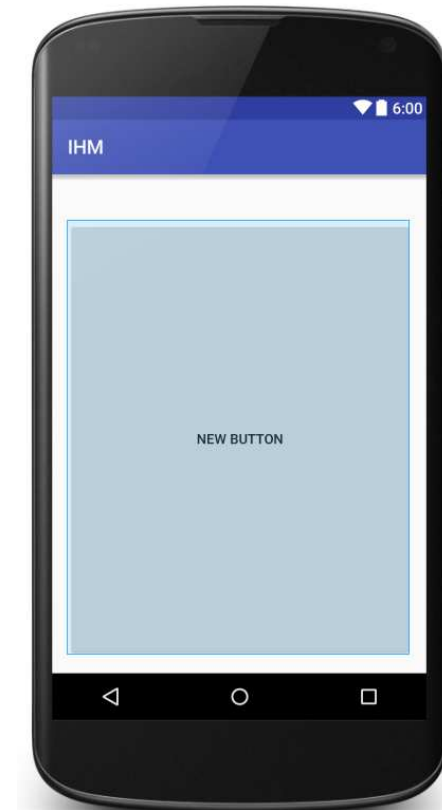
Second programme : une autre solution (5/5)

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

```
activity_main.xml x  app x  AndroidManifest.xml x
<?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

❑ bibliographie :

<http://developer.android.com/reference/android/view/View.html>
© JMF (Tous droits réservés)

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

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;

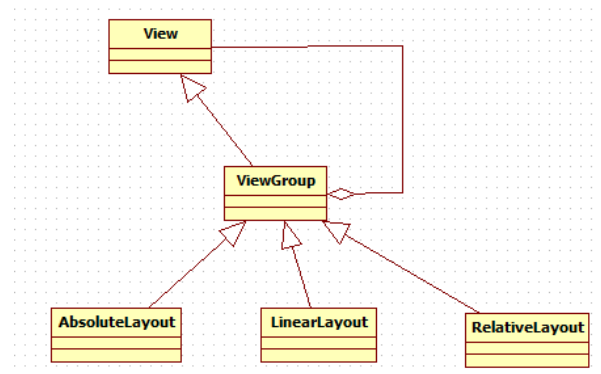
import java.util.Date;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

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 `XXLayout` !
- ❑ 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 conteneur AWT géré par un `FlowLayout` AWT)
 - ❑ `RelativeLayout`

❑ 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'exécution 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



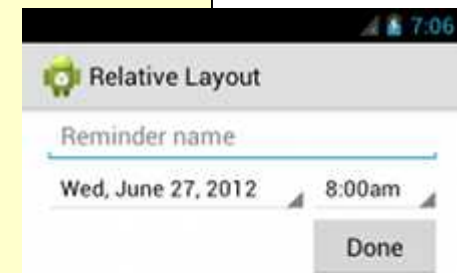
- ❑ = un conteneur qui permet de placer ses contenus les uns par rapport aux autres
- ❑ C'est le conteneur proposé dans le (nouveau) outil de construction d'IHM
- ❑ Les `Views` contenues dans le `RelativeLayout` indiquent leur positionnement à l'aide de leurs attributs (dans le fichier XML de l'IHM)
- ❑ 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="match_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>
```



@+id **vs.** @id

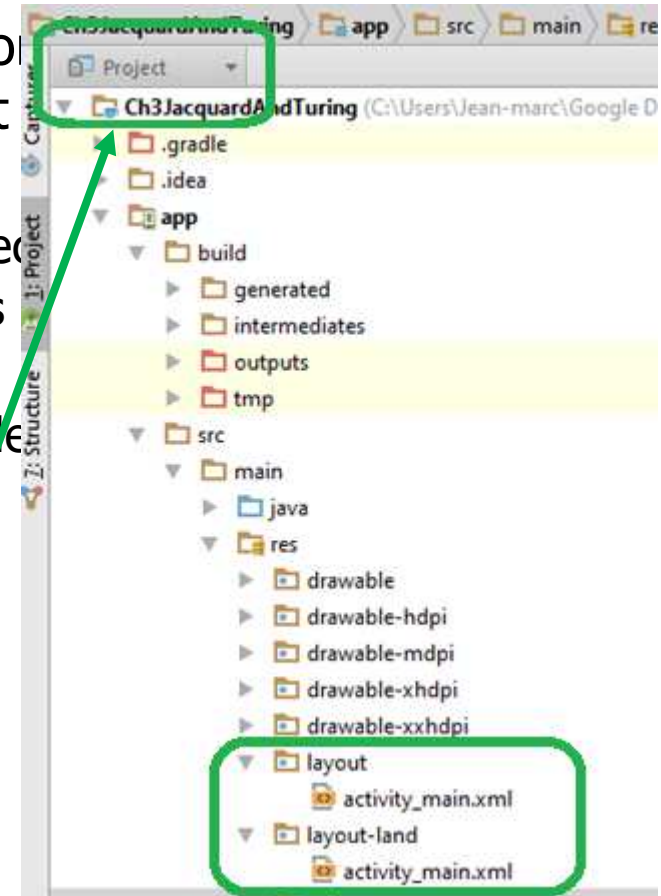
- ❑ Dans le fichier xml précédent, on utilise parfois @+id et parfois @id
- ❑ @+id indique qu'il faut créer, si nécessaire, une nouvelle entrée dans R.java (et sa classe interne id)
- ❑ @id repère simplement l'identificateur id et il n'y a pas de création dans R.java
- ❑ En fait cela fonctionne si on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

❑ Remarque : Sélectionner l'onglet Project pour cela, pas l'onglet Android



A propos de portrait-paysage (2/2)


- Ainsi, avec la seule activité :

on obtient

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

Jacquard and Turing


Alan Turing (1912 – 1954)



Alan Turing was a british scientist in the first part of XXth century. His contribution for the resolution of cryted nazy messages make the second world war finished earlier. Despite this, at the end of his life, he was worried for his "non conventional" life et died at only 41 years old. The Queen granted him a posthumous pardon on 24 December 2013 (less than one year ago)

Jacquard and Turing

Joseph Marie Jacquard




In a french point of view (so, may be it is not the truth ;-)), Joseph Marie Jacquard is one of the first computers inventor. He was born in Lyon (France) the 7 July 1752 and dead in Oullins (near Lyon) the 7 august 1834. So, he survive the french revolution (-:) I am sorry but there is no political remark or judgment here). He played an important role in the development of the earliest programmable loom (the Jacquard

Portrait-paysage : une démo

❑ demo Ch3JacquardAndTuring

Jacquard and Turing

Alan Turing (1912 – 1954)




Alan Turing was a british scientist in the first part of XXth century. His contribution for the resolution of cryted nazy messages make the second world war finished earlier.

Despite this, at the end of his life, he was worried for his "non conventional" life et died at only 41 years old. The Queen granted him a posthumous pardon on 24 December 2013 (less than one year ago).

Jacquard and Turing

Joseph Marie Jacquard



In a french point of view (so, may be it is not the truth ;-)), Joseph Marie Jacquard is one of the first computers inventor. He was born in Lyon (France) the 7 July 1752 and dead in Oullins (near Lyon) the 7 august 1834.

So, he survive the french revolution (-:) I am sorry but there is no political remark or judgment here). He played an important role in the development of the earliest programmable loom (the Jacquard

- ❑ 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, Layouts, etc

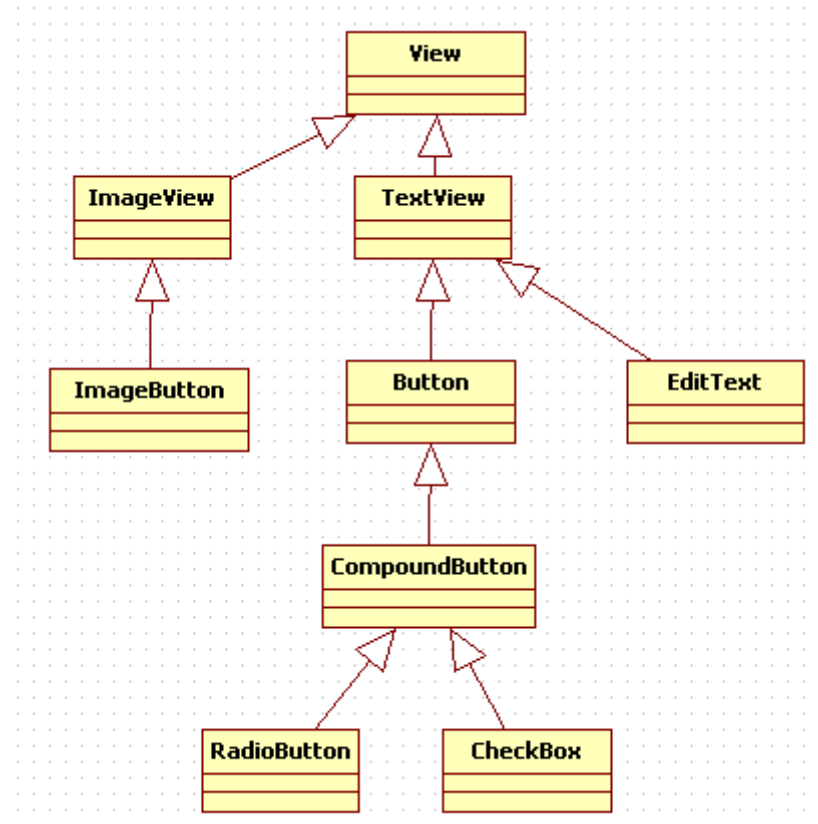
- ❑ "You can specify multiple qualifiers for a single set of resources, separated by dashes. For example, `drawable-en-rUS-land` applies to US-English devices in landscape orientation.
- ❑ The qualifiers must be in the order listed in table 2. For example:
Wrong: `drawable-hdpi-port/`
Correct: `drawable-port-hdpi/`
- ❑ 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 éditables (~ Label AWT) ou éditables (~ 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



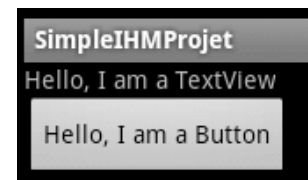
- ❑ Peut servir de zone de texte non éditable (~ `Label` AWT) et dans ce cas, sert souvent pour présenter les widgets qui suivent
- ❑ Propriétés importantes :
 - ❑ `android:text` : le texte du `TextView`
 - ❑ `android:typeface` : le type de police utilisée (monospace, ...)
 - ❑ `android:textStyle` : 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="match_parent"
    android:layout_height="match_parent"
    ...
    android:paddingTop="8dp" >

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

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 savedInstanceState) {
    ...
    // Récupération du Button à partir de l'IHM en XML
    Button button = (Button)findViewById(R.id.monBouton);
    // 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'une 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
}
```

Plus sur les Intents

- ❑ Un Intent est une description abstraite d'une opération à faire.
- ❑ Un Intent peut être utilisé avec :
 - ❑ `startActivity()` pour demander à lancer une activité
 - ❑ `broadcastIntent()` pour demander à contacter un `BroadcastReceiver`
 - ❑ `startService()` ou `bindService()` pour communiquer avec un `Service`
- ❑ Essentiellement un Intent est formé :
 - ❑ d'une description d'action à effectuer
 - ❑ de données utiles pour cette action
- ❑ source :
<https://developer.android.com/reference/android/content/Intent.html>

"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);
```

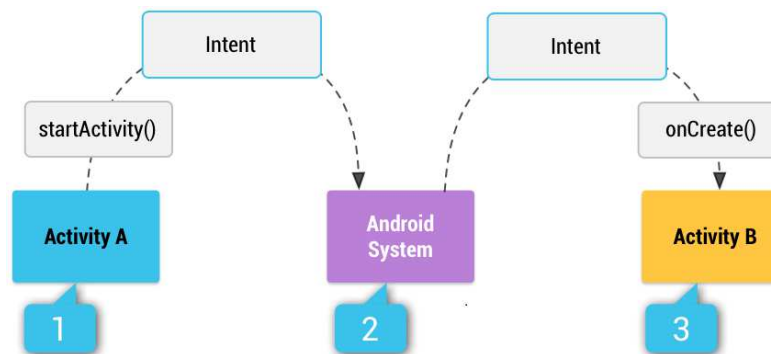
et déclarer la nouvelle activité `NouvelleActivity.class` (le futur écran) dans `AndroidManifest.xml`

- ❑ `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)

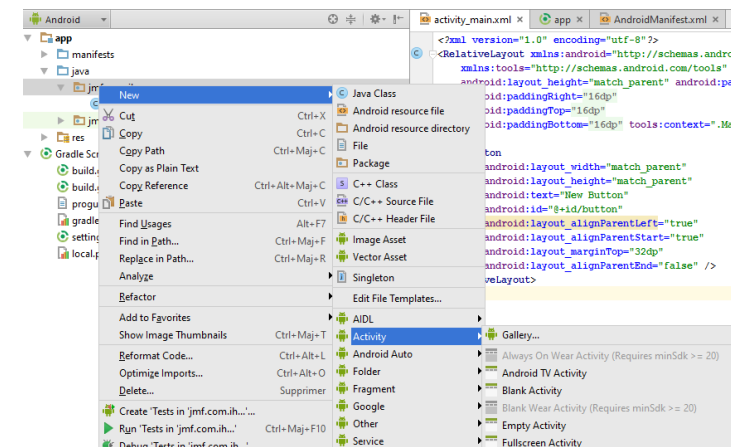
- ❑ En fait cet `Intent` est envoyé à l'environnement d'exécution. Celui-ci le redirige vers l'activité concernée



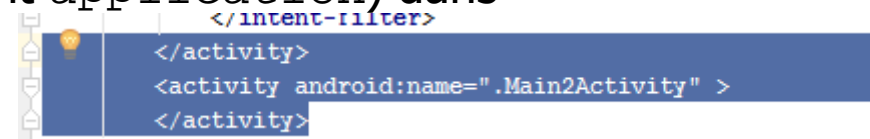
- ❑ Le premier argument du constructeur de l'`Intent` doit être le `Context`. Si l'appel est fait dans une activité `MonActivity`, `this` est souvent utilisé et convient car `Activity` dérive de `Context`
- ❑ On utilise souvent `MonActivity.this` quand on est dans un listener d'événement. Mais, la méthode `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 `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 `AndroidManifest.xml` aussi !



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 récupère 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 `getTypeEntrée(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);
```

```
ou : getIntent().getDoubleExtra("latitude",0);
```

La méthode

startActivityForResult()

- ❑ Permet de lancer une activité et pouvoir récupérer un résultat de cette nouvelle activité

- ❑ Le lancement est de la forme :

```
Intent lIntent = new Intent(getApplicationContext(), AutreActivity.class);
startActivityForResult (lIntent, CODE_DE_LA_REQUETE);
```

- ❑ CODE_DE_LA_REQUETE est un int identifiant la requête de lancement de la nouvelle activité
- ❑ La nouvelle activité va retourner des valeurs dans des extras de l'Intent retourné par :

```
Intent intentARetourner = new Intent();
intentARetourner.putExtra("valeurARetourner", resultatARetourner);
setResult(Activity.RESULT_OK, intentARetourner);
finish();
```

- ❑ Si tout se passe bien il faut utiliser RESULT_OK, sinon mettre RESULT_CANCELED
- ❑ biblio : <http://developer.android.com/training/basics/intents/result.html>

La méthode onActivityResult ()

- ❑ C'est cette méthode qui, en fait, permet récupérer un résultat de d'une autre activité
- ❑ Son code est de la forme :

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
    if ((resultCode == RESULT_OK) && requestCode == CODE_DE_LA_REQUETE) {  
        // traitement du résultat retourné  
        ... data.getXXXExtra("valeurARetourner")...  
    }  
}
```

- ❑ XXX et le type de la valeur retournée
- ❑ Remarque : C'est la technique pour passer des informations d'une activité appelée à une activité appelante

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) hiérarchisées. 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

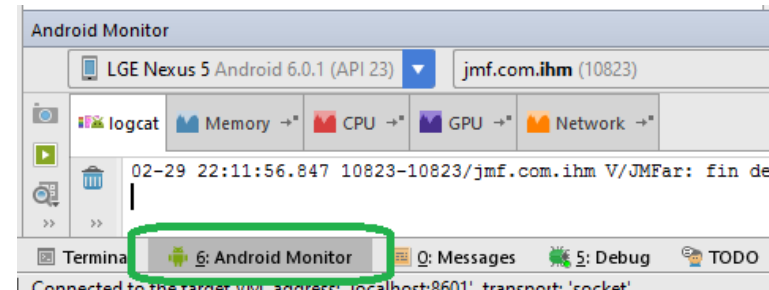
```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.new_game:
            Log.v("JMF", "Une nouvelle partie ?");
            Log.d("JMF", "Une nouvelle partie ?");
            Log.i("JMF", "Une nouvelle partie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

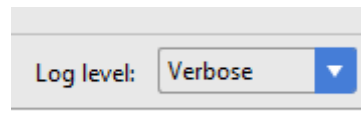


- ❑ Il y a alors plusieurs onglets disponibles utiles lors de l'exécution de l'app dont l'onglet logcat
- ❑ On peut faire des filtres sur les sorties dans la zone de texte de recherché de cet onglet



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

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 abréviations 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 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

Internationalisation en français

Que choisissez vous ?

☒ Des pommes

☐ Des poires

☐ Des scoubidous bidous

ou

English Internationalisation

What do you want ?

☒ Apples

☐ Pears

☐ Or scoubidous bidous (as Sacha sang)

car ...

Localisation : un exemple (2/2)

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

Internationalisation en français

Que choisissez vous ?

- ☒ Des pommes
- ☐ Des poires
- ☐ Des scoubidous bidous

English Internationalisation

What do you want ?

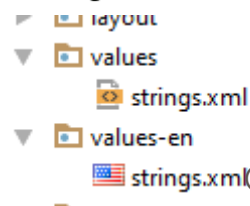
- ☒ Apples
- ☐ Pears
- ☐ Or scoubidous bidous (as Sacha sang)

```
<?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>
```

et 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 :



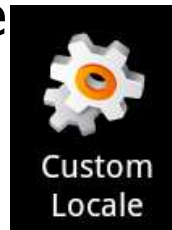
(utiliser l'onglet Project pas Android)

Configuration de la localisation

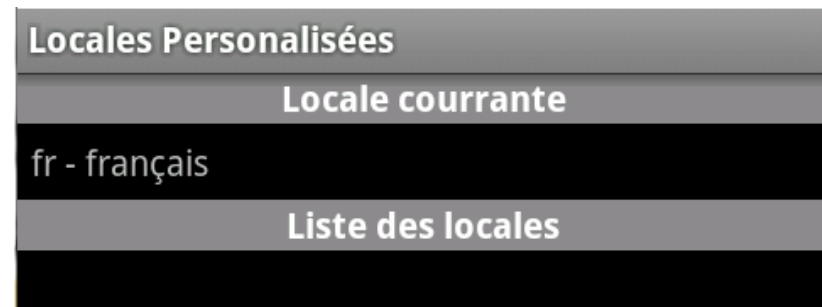
❑ Voir démo projet Ch3InternationalisationProjet

❑ 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>)

Taille : différence entre px, dp, dip, sp (1/3)

- ❑ Les tailles "physiques" :
 - ❑ px : pixel - correspond aux pixels (points élémentaires) de l'écran
 - ❑ in : inch : la taille en pouce (= unité de mesure de longueur).
1 pouce = 25,4 millimètres
 - ❑ mm : millimètre : la taille en millimètres (= unité de mesure de longueur)
 - ❑ pt : point : 1/72 de pouce
- ❑ source :
<http://stackoverflow.com/questions/2025282/difference-of-px-dp-dip-and-sp-in-android>

Taille : différence entre px, dp, dip, sp (2/3)

- ❑ Les tailles "logiques". Dans les faits, ce sont celles-ci qu'il faut utiliser
- ❑ dp = dip : Density-independent Pixels - an abstract unit that is based on the physical density of the screen. These units are relative to a 160 dpi screen, so one dp is one pixel on a 160 dpi screen. The ratio of dp-to-pixel will change with the screen density. The compiler accepts both "dip" and "dp", though "dp" is more consistent with "sp"
- ❑ En fait, $dp = (\text{resolution en dpi} / 160) \text{ px}$. Sur un écran de résolution de 160 dpi, $dp = \text{px}$ mais si la résolution est 320 dpi, on a $1 \text{ dp} = 2 \text{ pixels}$
- ❑ Conclusion : expression en dp => le système convertit cette unité en pixels physiques => l'image aura la même taille quel que soit l'écran

Taille : différence entre px, dp, dip, sp (3/3)

- ❑ sp : Scale-independent Pixels - this is like the dp unit, but it is also scaled by the user's font size preference. It is recommended you use this unit when specifying font sizes, so they will be adjusted for both the screen density and user's preferences
- ❑ Bref il faut utiliser sp à la place de dp lorsqu'il s'agit de taille de polices de caractères
- ❑ "To make it absolutely clear - try to never use anything but sp or dp unless you absolutely have to. Using sp/dp will make your Android applications compatible with multiple screen densities and resolutions. – Daniel Lew"

Styles et Thèmes

- ❑ Un style est un ensemble de propriétés précisant l'aspect de composants graphiques (`View`)
- ❑ Il est défini dans un fichier XML séparé de l'IHM
- ❑ Cette notion est similaire au CSS (Cascading Style Sheets) pour le web avec la même philosophie : séparer le contenu de l'aspect
- ❑ De même que pour les CSS, ces informations de style sont hiérarchisées
- ❑ Un thème est un style pour une `Activity` ou une application
- ❑ Une activity (resp. application) ayant un thème associé transmet les indications du thème à toutes les `View` contenues dans l'activité (resp. application)
- ❑ source :
<http://developer.android.com/guide/topics/ui/themes.html>

Séparer contenu et aspect avec les styles

❑ Au lieu d'écrire

```
<TextView  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:textColor="#00FF00"  
    android:typeface="monospace"  
    android:text="@string/hello" />
```

on écrira plutôt

```
<TextView  
    style="@style/CodeFont"  
    android:text="@string/hello" />
```

et les informations de style sont repérées par l'identificateur CodeFont

❑ CodeFont est en fait une entrée (i.e. un élément XML) dans un fichier de style

Fichier de styles

- ❑ Un fichier de styles est un fichier XML de nom quelconque rangé dans `res/values`. Son noeud racine doit être `resources`. Exemple

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="CodeFont" parent="@android:style/TextAppearance.Medium">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#00FF00</item>
    <item name="android:typeface">monospace</item>
  </style>
</resources>
```

- ❑ Chaque style est un élément XML `style`. Le nom du style est la valeur de l'attribut `name`. Il contient une suite de sous éléments `item` précisant une partie du style. Le corps de la balise `item` est la valeur de cet item
- ❑ Les styles peuvent être rangés par arborescence. On le précise par l'attribut (optionnel) `parent`

Héritage des styles

- ❑ Définir une nouvelle valeur pour un attribut de style écrase l'ancienne valeur (si elle existe) pour cet attribut
- ❑ On peut hériter de styles prédéfinis par Android et, dans ce cas, on utilise parent comme dans :

```
<style name="GreenText" parent="@android:style/TextAppearance">  
    <item name="android:textColor">#00FF00</item>  
</style>
```

- ❑ On peut hériter de ses propres styles et, dans ce cas, on utilise la notation . comme dans :

```
<style name="CodeFont.Red">  
    <item name="android:textColor">#FF0000</item>  
</style>
```

Le style défini est `CodeFont.Red`. Il récupère les valeurs du style `CodeFont`. On l'utilise avec la syntaxe `@style/CodeFont.Red`

- ❑ On peut créer une arborescence de style comme `CodeFont.Red.Big`

Les thèmes

- ❑ Un style appliqué à un `Viewgroup` ne se propage pas pour les composants contenus dans le `ViewGroup`
- ❑ Un thème appliqué à une activité est propagé à tous les composants de l'activité
- ❑ Les indications de thème sont écrits dans le `AndroidManifest.xml` à l'aide l'attribut `android:theme` de la balise `application` :

```
<application android:theme="@style/CodeFont">
```

- ❑ On peut utiliser les thèmes prédéfinis d'Android :

```
<activity android:theme="@android:style/Theme.Dialog">
```

- ❑ Les styles et thèmes proposés par Android sont accessibles à partir de <http://developer.android.com/guide/topics/ui/themes.html#PlatformStyles>

Résumé du chapitre (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. On utilise, pour cela les `Intents`

Résumé du chapitre (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
- ❑ Afin de dissocier le contenu à afficher et la présentation de ce contenu on utilise les styles et les thèmes