



IN01

Programmation Android

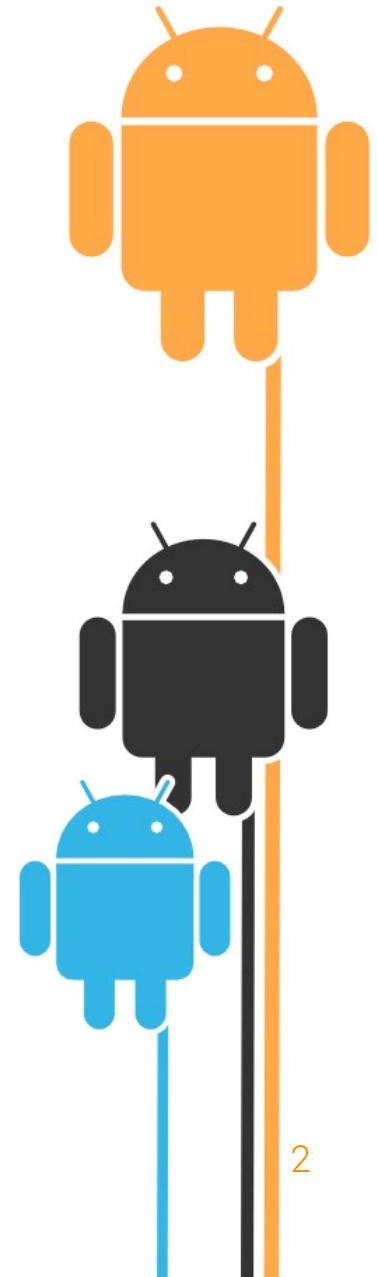
03 – IHM bases

Yann Caron
Et Jean-Marc Farinone

ENSG
Géomatique

Sommaire - Séance 03

- Principes et patrons
- Combinaison ou XML
- Évènements
- Standard views
- Menus
- Intents et chaînage des activities
- Toast



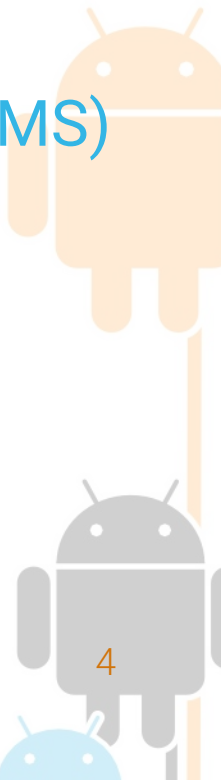
IN01 – Séance 03

Principes et patrons



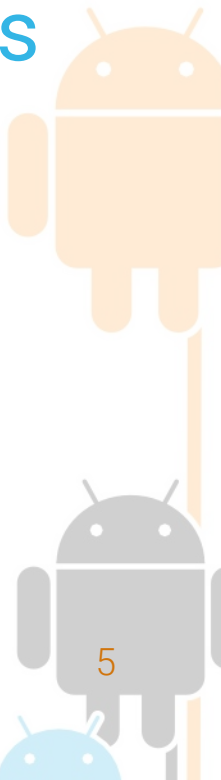
Les IHM des smartphones

- Pas seulement clavier écran souris
- Mais écran tactile, clavier virtuel
- Accéléromètre
- GPS
- Enregistrement de messages audio (notes audio)
- Prendre des photos, des vidéos, les envoyer par le réseau (MMS)
- SMS
- Téléphone
- Etc.



Les IHM Android

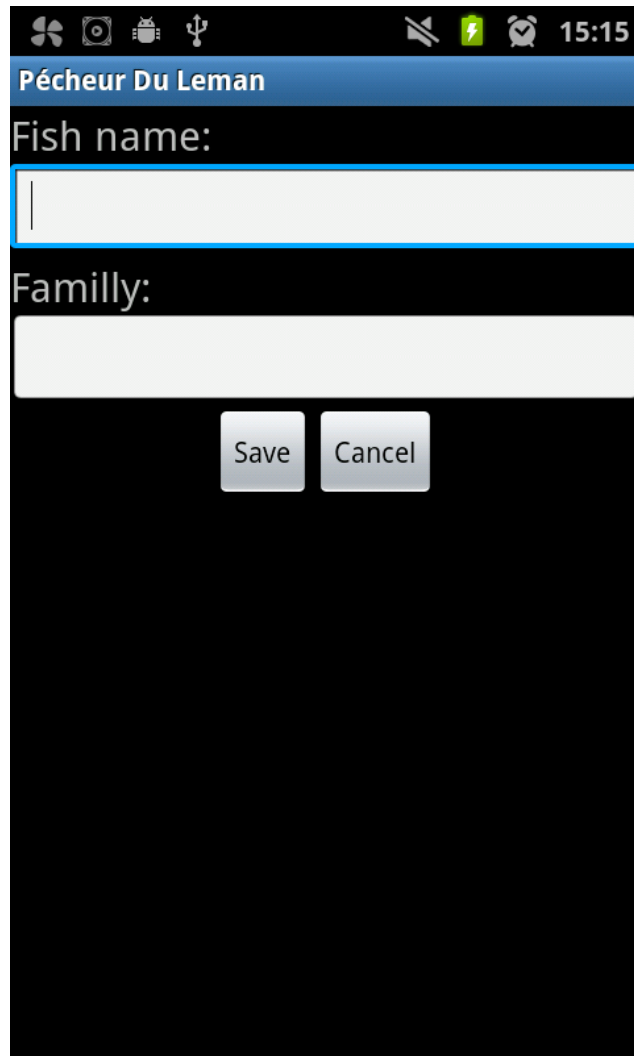
- Bibliothèque propre
- Pas d'AWT, ni Swing, ni Java ME/LCDUI
- Décrit par fichier XML ou par composition
- Écrans gérés par des activities, des fragments et des views



Principes

- Deux grands principes == Deux arborescences
 - ➔ Par combinaison
 - Les IHM sont construites en mettant les composants graphiques les uns à l'intérieur des autres
 - Placement des composants
 - ➔ Par héritage
 - Les composants graphiques sont obtenus par spécialisation
 - En général fournis par le concepteur de la bibliothèque
 - Utile pour créer des composants personnalisés (IDE Algoid)

1^{er} principe : combinaison

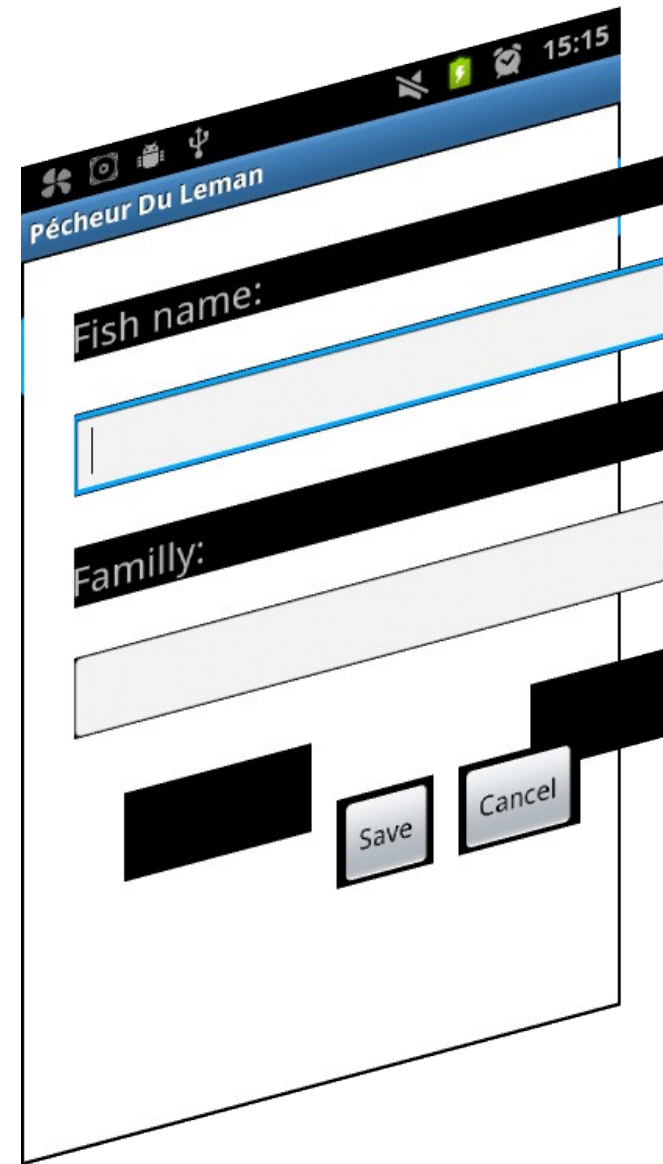


Pêcheur Du Lemman

Fish name:

Familly:

Save Cancel



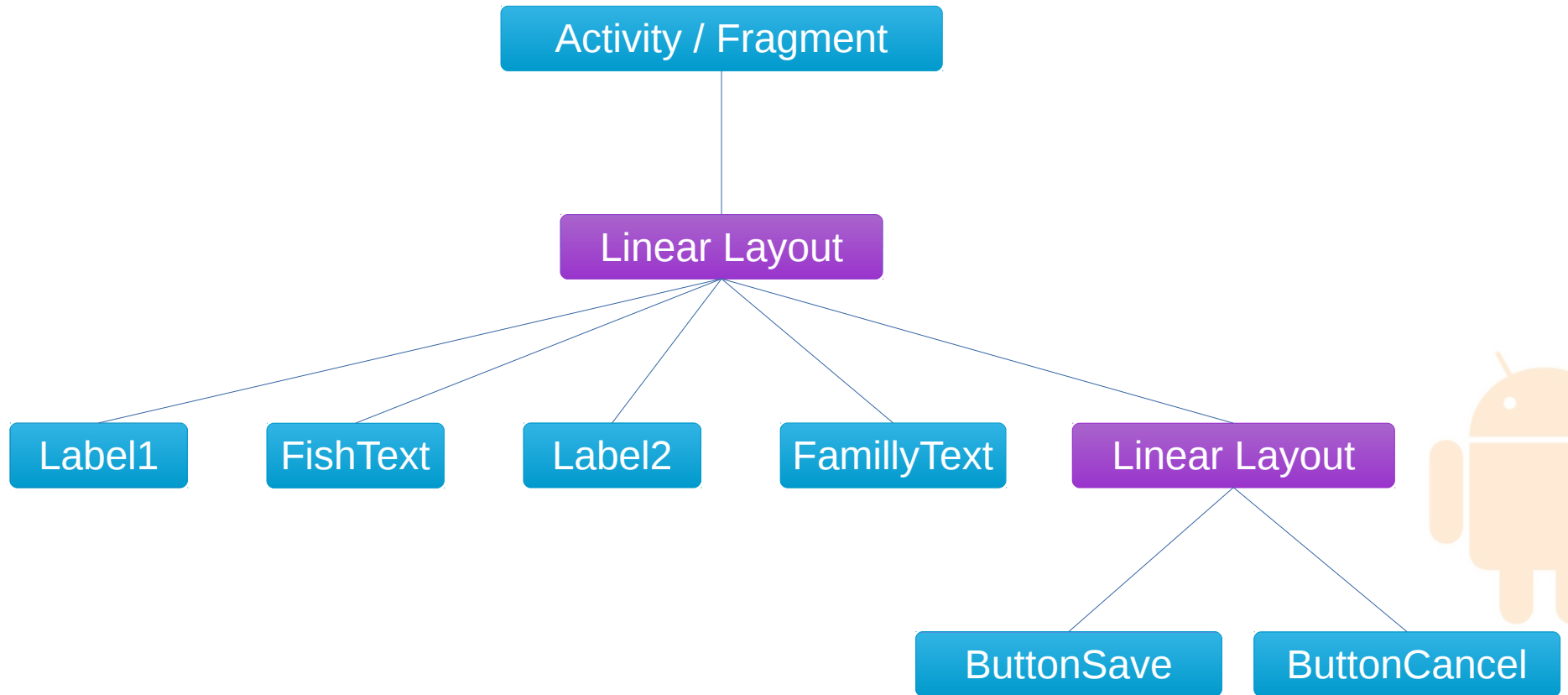
Pêcheur Du Lemman

Fish name:

Familly:

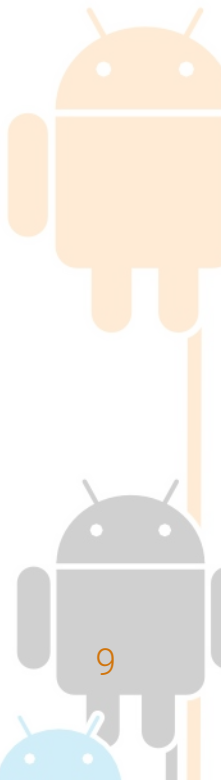
Save Cancel

1^{er} principe : combinaison



1^{er} principe : combinaison

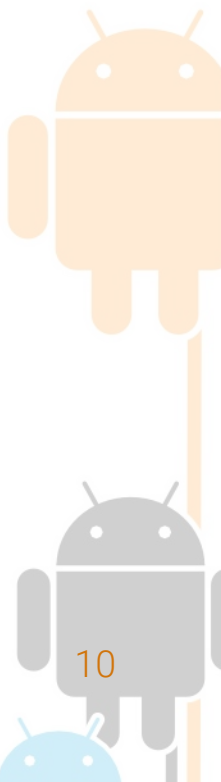
- Principe : mettre les composants les uns à l'intérieur des autres (hiérarchiser, combiner)
- Components et Containers
- On obtient un arbre : être contenu par == être fils de
- Deux patrons : composite et decorator



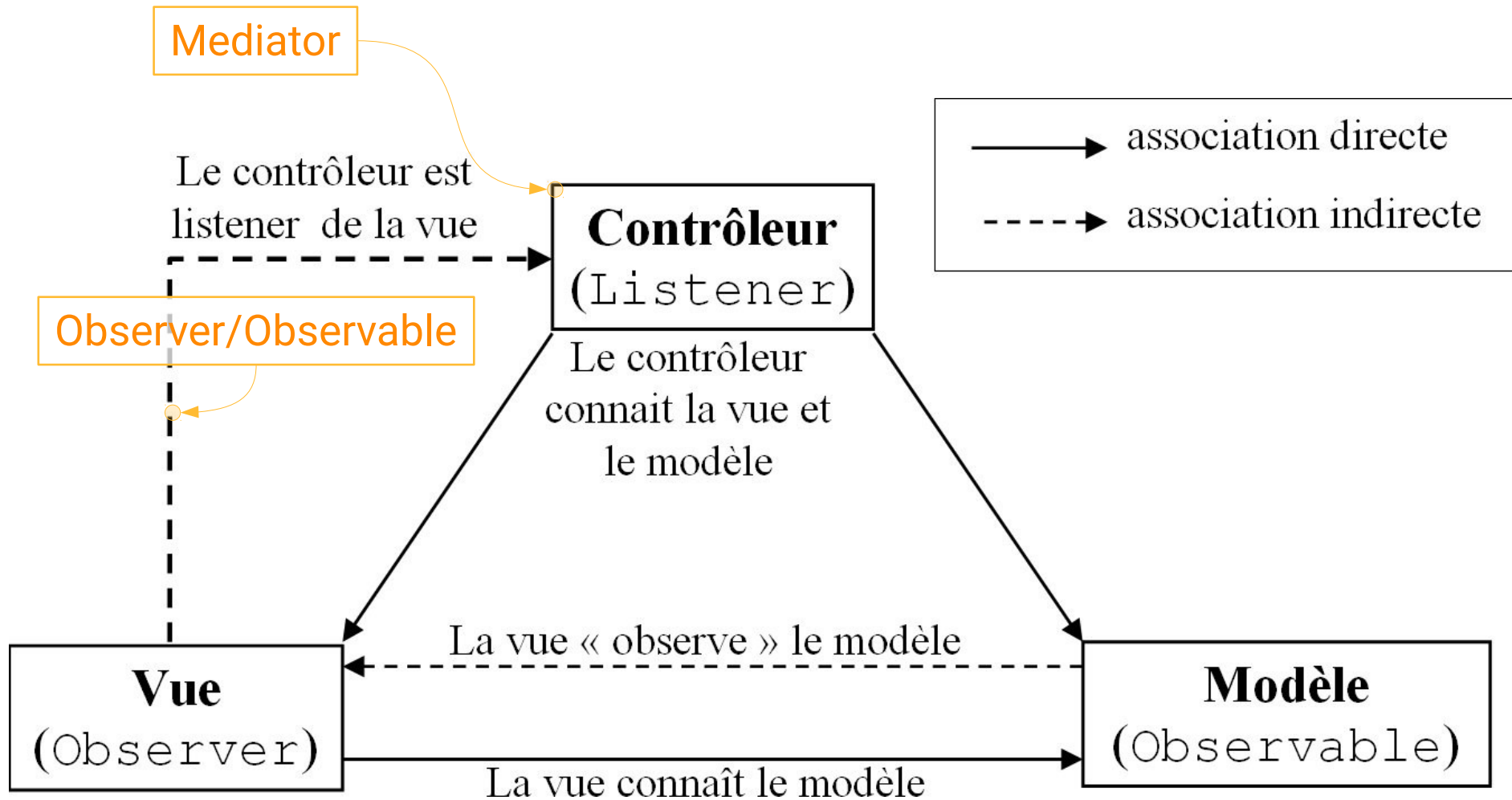
2^e principe – Héritage de classes

- Spécialisation des classes par héritages successifs
- Par exemple : MultiAutoCompleteTextView

```
java.lang.Object
↳ android.view.View
↳ android.widget.TextView
↳ android.widget.EditText
↳ android.widget.AutoCompleteTextView
↳ android.widget.MultiAutoCompleteTextView
```



Patron - MVC



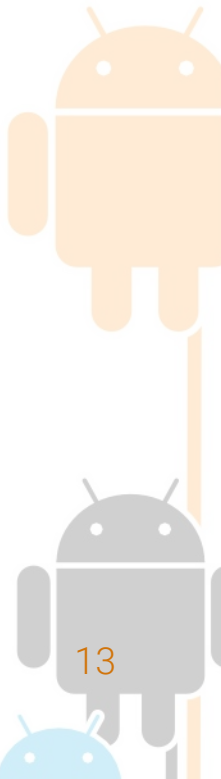
Patron - MVC

- POA : Pattern-Oriented Software Architecture
- Motivation : séparer les responsabilités d'un composant graphique
- Le modèle : c'est la donnée
- La vue : c'est la représentation visuelle
- Le contrôleur : c'est le chef d'orchestre (spécialisation du mediator)
- Un excellent article :
<http://www.infres.enst.fr/~hudry/coursJava/interSwing/boutons5.html>



MVC - Android

- Comme vu précédemment
MVC = Model/View/Controller
- L'idée est de séparer les données, la présentation et les traitements
- Les composants graphiques Android sont en partie basés sur ce principe



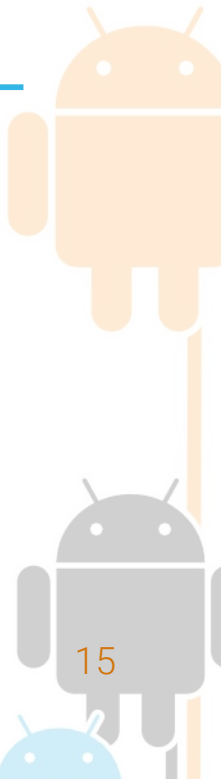
IN01 – Séance 03

Combinaison ou XML



IHM : types de constructions

- Deux façons de construire une IHM dans Android :
 - ➔ En combinant les composants par programmation
 - Comme JavaSE Swing, AWT, JavaME
 - ➔ En déclarant les composants dans un fichier XML
 - Comme JavaFX (FXML), WPF (XAML)



Par combinaison

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

    // create main layout
    LinearLayout content = new LinearLayout(this);
    content.setLayoutParams(
        new LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.MATCH_PARENT));
    content.setOrientation(LinearLayout.VERTICAL);

    // add main layout to activity
    this setContentView(content);

    // add controls
    TextView label1 = new TextView(this);
    label1.setText(getText(R.string.fish_name));
    content.addView(label1);

    EditText textFish = new EditText(this);
    textFish.setLayoutParams(
        new LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.WRAP_CONTENT));
    content.addView(textFish);

    // etc.
}
```

onCreate de l'activity

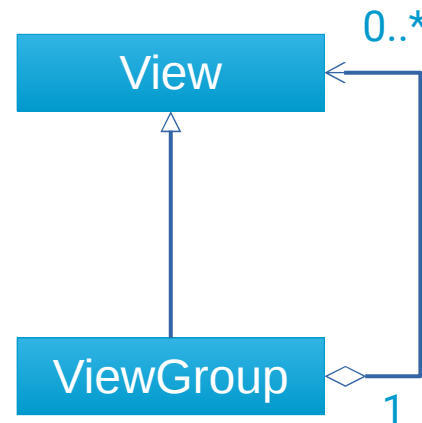
Content layout

Layout

Ajout des contrôles

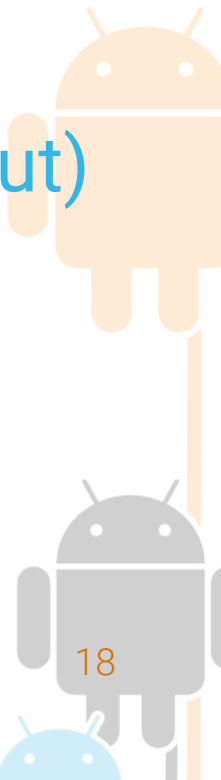
Par combinaison

- Comment ? Comme en Swing !
 - ➔ On instancie des objets
 - ➔ On les paramètre à l'aide des propriétés
 - ➔ On les hiérarchise grâce à un patron composite ()



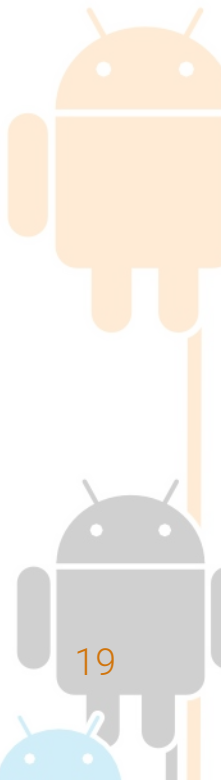
Par combinaison

- Redite :
- L'IHM est décrite par programmation en surchargeant la méthode “onCreate” de l'activity (ou du fragment)
- On instancie des composants graphiques (LinearLayout, TextView, Button, etc.)
- On les paramètre (setText, setOrientation, setLayout)
- On hiérarchise (content.addView) Content est le layout auquel on ajoute une vue



Par combinaison

- Avantages :
 - Dynamique (construction de vues par programmation)
 - Peut être factorisé
- Inconvénients :
 - Verbeux !
 - Fastidieux
 - WYSIWYG difficile à mettre en œuvre



Déclaration par XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >

  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/bait_name" />

  <AutoCompleteTextView
    android:id="@+id/autoCompleteTextView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="AutoCompleteTextView" >

    <requestFocus />
  </AutoCompleteTextView>

  <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal" >

    <Button
      android:id="@+id/button1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/global_button_save" />

  </LinearLayout>
</LinearLayout>
```

Imbrication

Pêcheur Du Lemman

Bait name:

Brand:

Price (\$):

Size (in):

Weight (oz):

Save Cancel

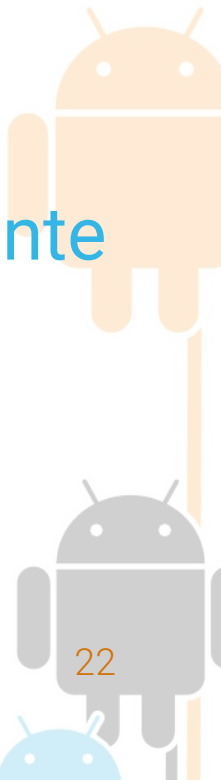
Déclaration par XML

- Le placer dans `/res/layout` (cf. Cours 02 pour la gestion multirésolution, orientation, etc.)
- La description d'une activity doit commencer par un `Layout`
- Les composants graphiques sont ensuite hiérarchisés par encapsulation dans les nœuds XML
- Les composants sont paramétrés avec des attributs XML



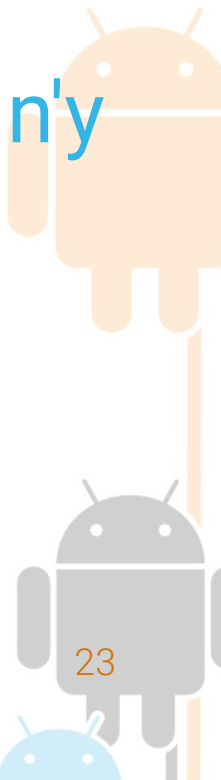
Déclaration par XML

- Avantages :
 - WYSIWYG plus simple à mettre en place
 - Plus lisible
- Inconvénients :
 - Limitations / bogues du WYSIWYG
 - En mode text, autocompletion pas toujours présente
 - Verbeux



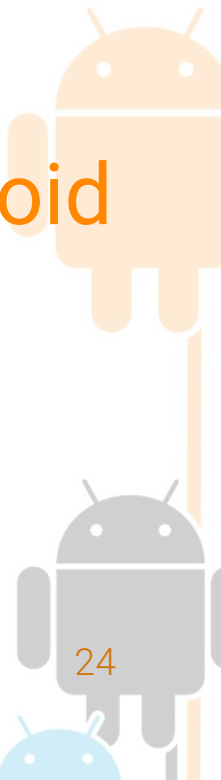
Post-Scriptum - @+id vs. @id

- Dans le fichier XML, on utilise parfois @+id et parfois @id
- @+id indique qu'il faut créer 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



Correspondance XML-Java

- Comme vu précédemment, il est possible de décrire les vues de deux façons
- Toutes les méthodes Java ont un équivalent sous forme d'un attribut XML
- La référence se trouve ici :
<http://developer.android.com/reference/android/view/View.html>



Correspondance XML-Java

- Dont voici un extrait

| XML Attributes | | |
|---------------------------------|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| Attribute Name | Related Method | Description |
| android:accessibilityLiveRegion | setAccessibilityLiveRegion(int) | Indicates to accessibility services whether the user should be notified when this view changes. |
| 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. |

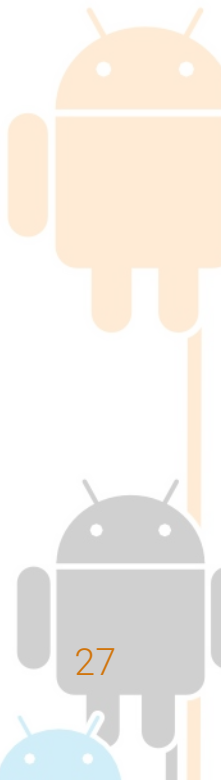
Activity et XML

```
public class BaitActivity extends Activity {  
  
    private autoCompleteTextView textName, textBrand;  
    private Button buttonSave;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.bait);  
  
        textName = (AutoCompleteTextView) findViewById(R.id.textName);  
        textBrand = (AutoCompleteTextView) findViewById(R.id.textBrand);  
  
        buttonSave = (Button) findViewById(R.id.buttonSave);  
  
        buttonSave.setOnClickListener(new OnClickListener() {  
  
            @Override  
            public void onClick(View v) {  
                buttonSave_onClick(v);  
            }  
        });  
    }  
  
    // event handling  
    private void buttonSave_onClick(View v) {  
        // do some stuff  
    }  
}
```

Accès aux contrôles

Activity et XML

- Bonnes pratiques :
 - ➔ Nommer correctement les composants de l'IHM (conventions de nommage)
 - ➔ Utiliser ces noms pour créer des attributs de la classe activity (ou fragment)
 - ➔ Récupérer leur instance lors de l'exécution de la méthode onLoad
 - ➔ Abonner des évènements si nécessaire



Eclipse-WYSIWYG

The screenshot displays the Eclipse IDE's WYSIWYG editor for an Android application. The central area shows a graphical layout for a form titled "Pêcheur Du Leman". The form contains several text input fields labeled "Bait name:", "Brand:", "Price (\$):", "Size (in):", and "Weight (oz):". At the bottom of the form are two buttons labeled "Save" and "Cancel".

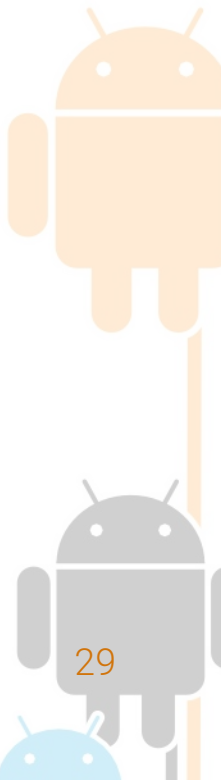
On the left, the **Palette** view shows a list of **Form Widgets** including TextView, Button, CheckBox, RadioButton, and various ProgressBar and SeekBar options. An orange arrow points from the **Drag and Drop** annotation to a widget in the palette.

On the right, the **Outline** view shows a tree structure of the UI components, including LinearLayout, TextView, EditText, and Button. An orange arrow points from the **Outline** annotation to this view.

At the bottom right, the **Properties** view shows the attributes of the selected component (a Button). It includes properties like Id, Layout Parameters, Margins, Style, Text, Hint, Content Description, and various text attributes. An orange arrow points from the **Propriétés** annotation to this view.

Eclipse-WYSIWYG

- Ajout des éléments par drag and drop
- Outline permet de gérer l'arborescence des composants graphiques
- Modification des propriétés dans la fenêtre “properties”

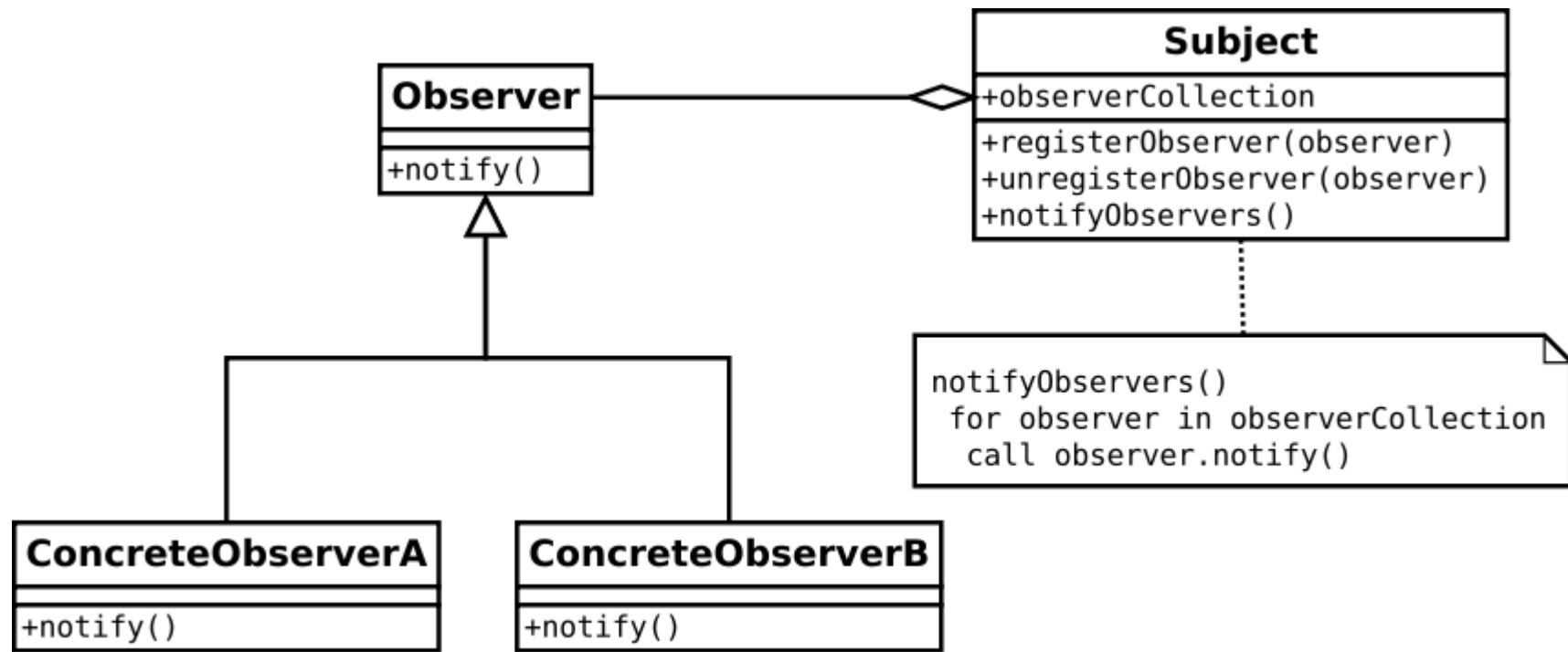


IN01 – Séance 03

Évènements

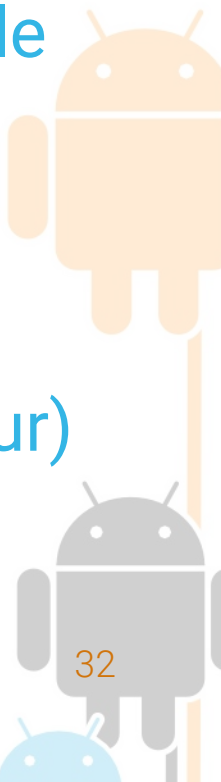


Patron – Observer/Observable



Patron – Observer/Observable

- Motivation : gestion des évènements
- L'observateur est une interface qui possède une méthode
- L'observateur s'abonne à l'observable (ce dernier stocke la référence des observateurs dans une collection)
- Lorsque l'observable déclenche son évènement, il boucle sur les références et invoque les méthodes de celles-ci
- Principe du callback
- Sur Android, ce sont parfois des observateurs multiréférences ou monoréférences (un seul observateur)



Patron – Observer/Observable

■ Multi-callbacks

```
public interface Observer {  
    void callBack();  
}  
  
public class Observable {  
    private final List<Observer> observers = new ArrayList<Observer>();  
  
    public void addObserver(Observer observer) {  
        observers.add(observer);  
    }  
  
    private void fireEvent() {  
        for (Observer ob : observers) {  
            ob.callBack();  
        }  
    }  
}
```

Diagram annotations:

- Interface**: Points to the `Observer` interface.
- Agrégation**: Points to the `observers.add(observer)` line in the `addObserver` method.
- Multi-callback**: Points to the `for` loop in the `fireEvent` method.

Patron – Observer/Observable

- Mono-callbacks

```
public interface Observer {  
    void callBack();  
}  
  
public class Observable {  
    private Observer observer;  
  
    public void setObserver(Observer observer) {  
        this.observer = observer;  
    }  
  
    private void fireEvent() {  
        observer.callBack();  
    }  
}
```

Diagram illustrating the Observer/Observable pattern with Mono-callbacks:

- Interface**: Points to the `Observer` interface.
- Agrégation**: Points to the `setObserver` method in the `Observable` class, indicating that `Observable` aggregates the `Observer` interface.
- Mono-callback**: Points to the `callBack` method call within the `fireEvent` method in the `Observable` class, indicating that there is only one callback method used for communication.

Patron – Observer/Observable

■ Exemple

```
LinearLayout container = (LinearLayout) findViewById(R.id.container);

Button button = new Button(this);
button.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        Log.i(this.getClass().getName(), "Button clicked !");
    }
});

container.addView(button);
```

Classe anonyme

Les évènements

- Les composants émettent des évènements (patron observer/observable)
- Pour s'y abonner, deux possibilités :
 - ➔ L'activité implémente l'interface de l'observateur (le listener)
 - Pas conseillé ! S'il y a plusieurs objets de même nature (bouton par exemple) comment fait-on ?
 - ➔ On passe l'instance d'une classe anonyme en paramètre. Sa méthode sera exécutée lors de l'activation de l'évènement
 - Une bonne pratique, ne pas mettre le code directement dans la méthode de la classe anonyme, mais on délègue à une méthode de l'activity



Évènements en Java - Exemple

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

    // manage event
    buttonSave.setOnClickListener(new OnClickListener() {

        @Override
        public void onClick(View v) {
            buttonSave_onClick(v);
        }
    });

    // event handling
    private void buttonSave_onClick(View v) {
        // do some stuff
    }
}
```

Classe anonyme

Délègue à une méthode

Évènements – En XML

- Mode déclaratif
- Attribut :
`android:onClick`
- Attention : pas de check du compilateur !

XML

```
<Button  
  android:id="@+id/buttonSave"  
  android:layout_width="wrap_content"  
  android:layout_height="wrap_content"  
  android:onClick="buttonSave_onClick"  
  android:text="@string/save" />
```

Nom de l'évènement

Activity - Java

```
// event handling  
private void buttonSave_onClick(View v)  
{  
    // do some stuff  
}
```

IN01 – Séance 03

Standard views

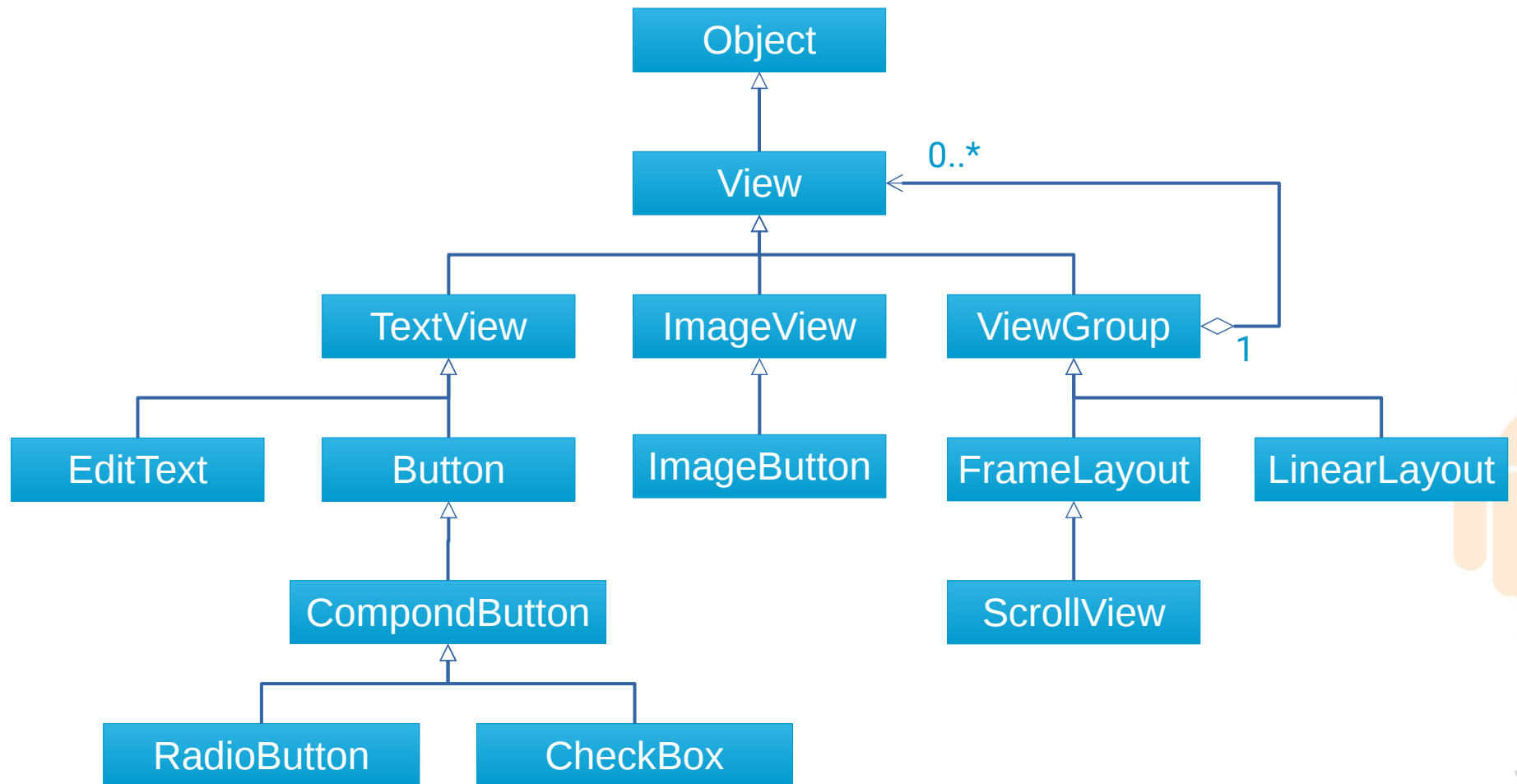


Les composants graphiques

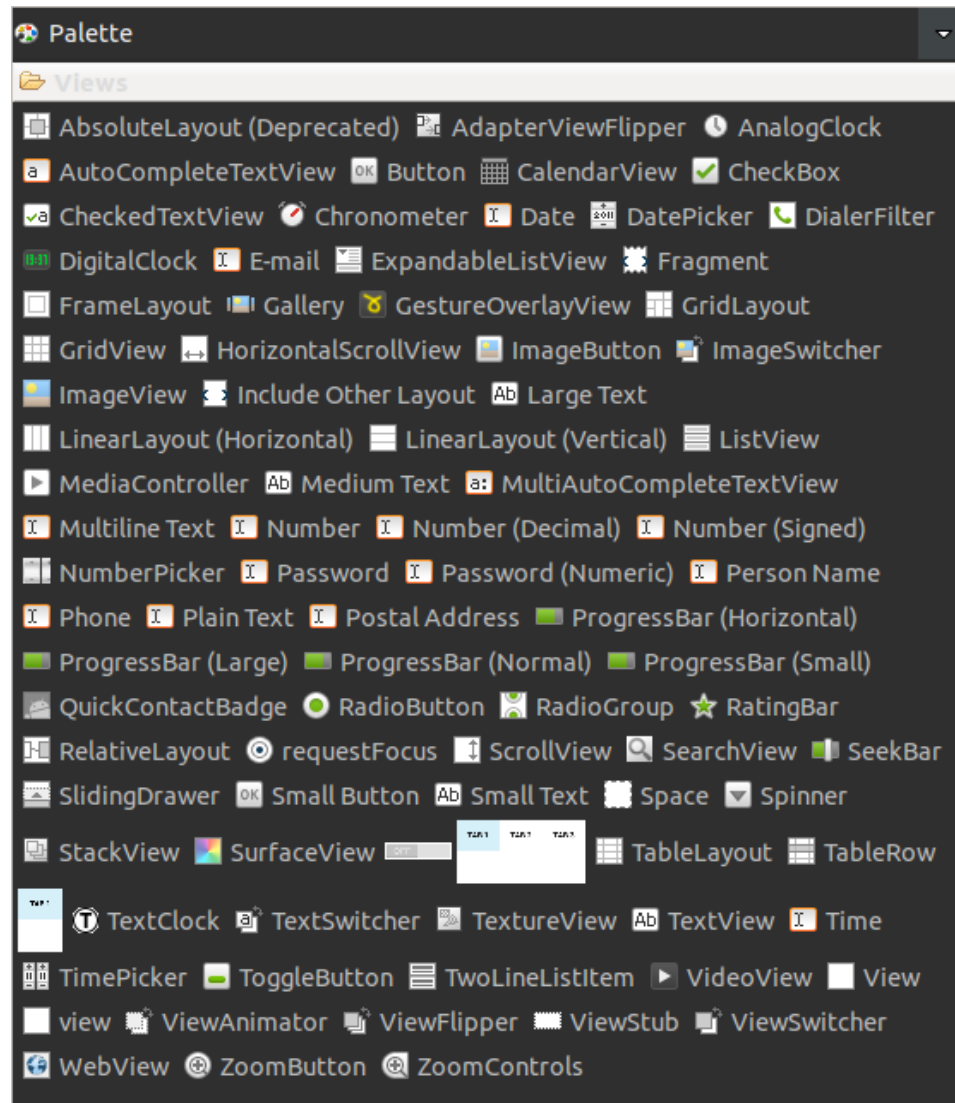
- Ce sont les composants graphiques que voit l'utilisateur, avec lesquels il agit sur (contrôle) l'interface graphique
 - ➔ Appelés dans certains domaines, les contrôles
 - ➔ En Android ce sont (par exemple) :
 - les zones de texte non éditables (~Label AWT) ou éditables (~TextComponent AWT) : `TextView`
 - ➔ 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`



Relation d'héritage



Vue d'ensemble



TextView

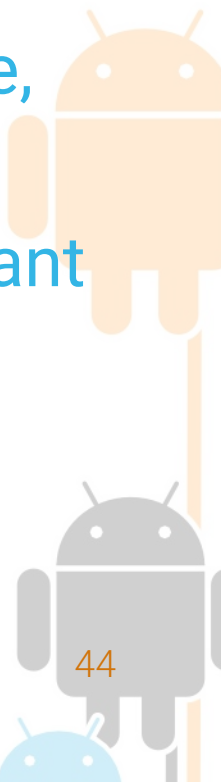
Price (\$):

- Sert de zone de texte non éditable (~ Label AWT)
- 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 hexadécimale en unités RGB (par exemple #FF0000 pour le rouge)

EditText

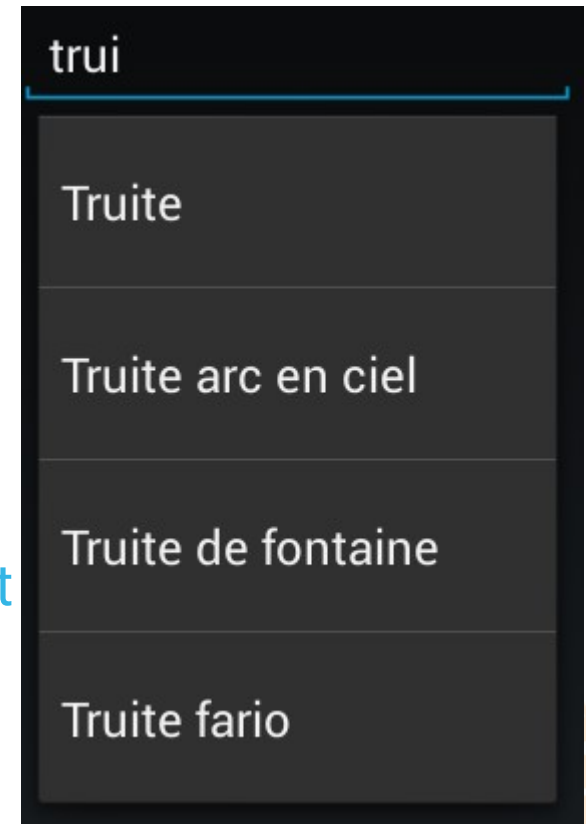


- Sert de zone de texte éditable (~ `TextField` AWT)
- Propriétés importantes :
 - ➔ `android:text` : idem `TextView`
 - ➔ `android:inputType` : spécifie le type d'entrée attendu (`numberDecimal`, `numberSigned`, `phone`, `date`, `textEmailAddress`, `textMultiLine`, `textPersonName`)
 - ➔ `android:hint` : texte à afficher lorsque le composant est vide



AutoCompleteTextView

- Entre la saisie de texte et la dropdown
- Propriétés importantes :
 - ➔ `android:completionThreshold` : nombre de caractères avant le déclenchement de la dropdown
- Méthodes importantes :
 - ➔ `setAdapter()` : charge la liste des données utilisées pour l'autocomplétion. La liste doit être filtrable.
- Objet important :
 - ➔ `ArrayAdapter` : permet d'adapter un tableau d'objets (`String` en général, ou `toString()`)



AutoCompleteTextView - Exemple

```
textName = new AutoCompleteTextView(this);
```

```
textName.setAdapter(  
    new ArrayAdapter<String>(   
        this,  
        android.R.layout.simple_dropdown_item_1line,  
        new String[] {"Brochet", "Perche", "Truite"}));
```

```
content.addView(textName);
```

Context

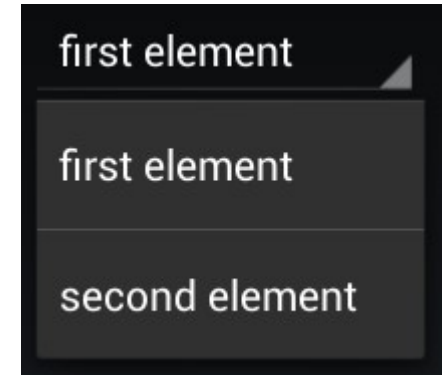
Composant utilisé
pour afficher le
text

Liste de la dropdown

- Remarque : l'IDE d'Algoid est entièrement basé sur un **MultiAutoCompleteTextView**. Ce dernier propose une drop down pour chaque mot du texte (spécifier un séparateur, comme l'espace pour du code source)

Spinner

- Une combobox sans saisie (~ CheckBox AWT)
- Propriétés importantes :
 - ➔ `android:prompt` : indicateur textuel lorsque la liste s'affiche
 - ➔ `android:entries` : spécifie les éléments de la liste (doit faire référence à un tableau)



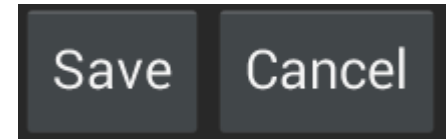
```
<string-array name="my_array">  
  <item>first element</item>  
  <item>second element</item>  
</string-array>
```

Spinner

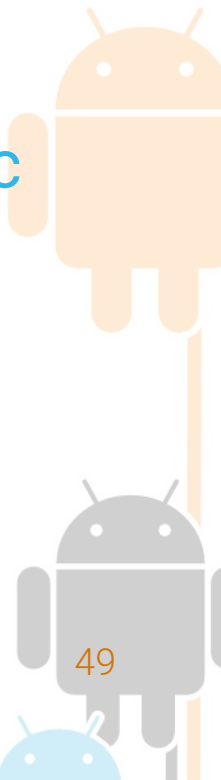
- Comme l'AutoCompleteTextView il peut utiliser un adapter
- Méthodes importantes :
 - ➔ setAdapter : charge la liste des données

```
Spinner spinner = (Spinner) rootView.findViewById(R.id.autoCompleteTextView1);  
spinner.setAdapter(  
    new ArrayAdapter<String>(rootView.getContext(),  
        android.R.layout.simple_dropdown_item_1line,  
        new String[]{"Ablette", "Brochet", "Féra", "Gardon", "Omble chevalier"}));
```

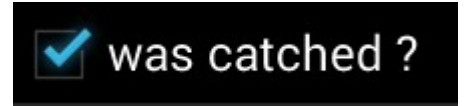

Button



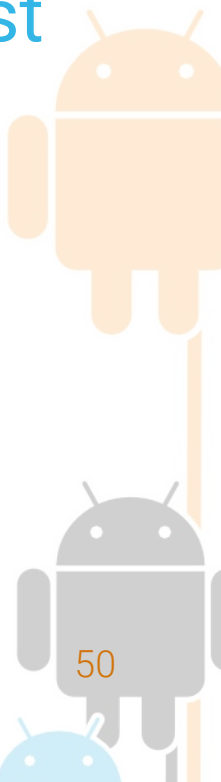
- Un bouton (~ Button AWT)
- Propriétés importantes :
 - ➔ `android:text` : idem TextView
 - ➔ `android:onClick` : abonne un listener sur un évènement clic de l'utilisateur
 - ➔ `android:drawableLeft` : une image à afficher avec les textes à la position souhaitée (Top, Bottom, Left, Right)
 - ➔ `android:enable` : actif ou inactif



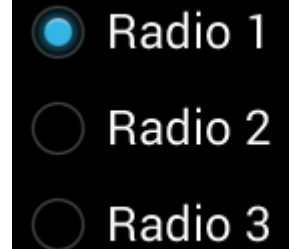
CheckBox



- Une case à cocher (~ CheckBox AWT)
- Propriétés importantes :
 - ➔ `android:text` : idem `TextView`
 - ➔ `android:checked` : spécifie si le composant est coché par défaut



RadioGroup et RadioButton



- Un groupe de boutons dépendants
(~ CheckBox & CheckboxGroup AWT)
- Pour que les RadioButton soient dépendants, il faut qu'ils soient contenus dans le même groupe

```
<RadioGroup
    android:id="@+id/radioGroup1">

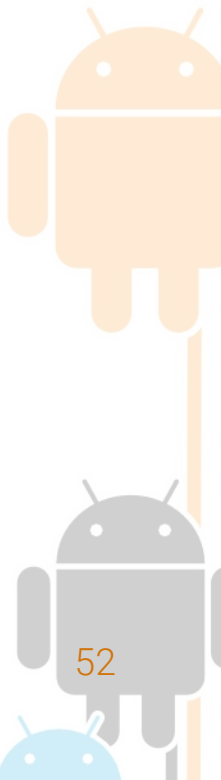
    <RadioButton
        android:id="@+id/radio0"
        android:checked="true"
        android:text="@string/radio1" />

    <RadioButton
        android:id="@+id/radio1"
        android:text="@string/radio2" />

    <RadioButton
        android:id="@+id/radio2"
        android:text="@string/radio3" />
</RadioGroup>
```

RadioButton et évènements

- Deux options :
 - ➔ Créer un listener par RadioButton
 - ➔ Utiliser la même méthode pour tous les RadioButton d'un même RadioGroup. Dans la méthode on sait quel RadioButton appelle grâce à l'attribut View



RadioButton et évènements

XML

```
<RadioGroup
    android:id="@+id/radioGroup1">

<RadioButton
    android:id="@+id/radio0"
    android:checked="true"
    android:onClick="radio_onClick" />

<RadioButton
    android:id="@+id/radio1"
    android:onClick="radio_onClick" />

<RadioButton
    android:id="@+id/radio2"
    android:onClick="radio_onClick" />

</RadioGroup>
```

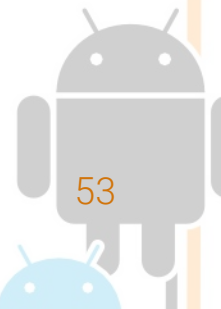
Activity - Java

```
OnClickListener listener = new
OnClickListener() {
    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
    }
};

RadioGroup rg1 = (RadioGroup)
findViewById(R.id.radioGroup1);



for (int i = 0; i < rg1.getChildCount(); i++) {
    RadioButton r =
        (RadioButton) rg1.getChildAt(i);

    r.setOnClickListener(listener);
}
```



RadioButton et évènements

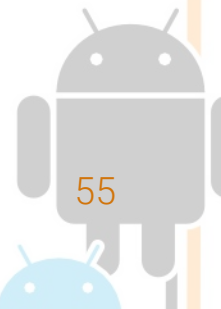
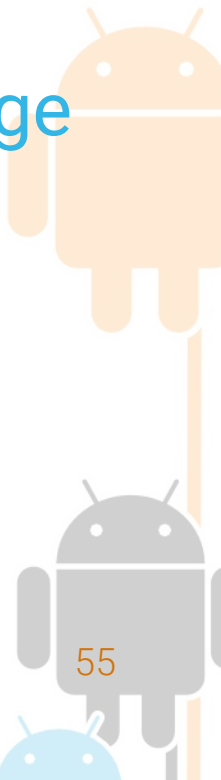
- On teste quel bouton radio est à l'origine de l'évènement

```
private void radio_onClick(View v) {  
    RadioButton radio = (RadioButton)v;   
    Log.i(this.getClass().getName(), radio.getId()+" is clicked");  
  
    switch (radio.getId()) {   
        case R.id.radio1:  
            break;  
        case R.id.radio2:  
            break;  
        default:  
    }  
}
```

ImageView

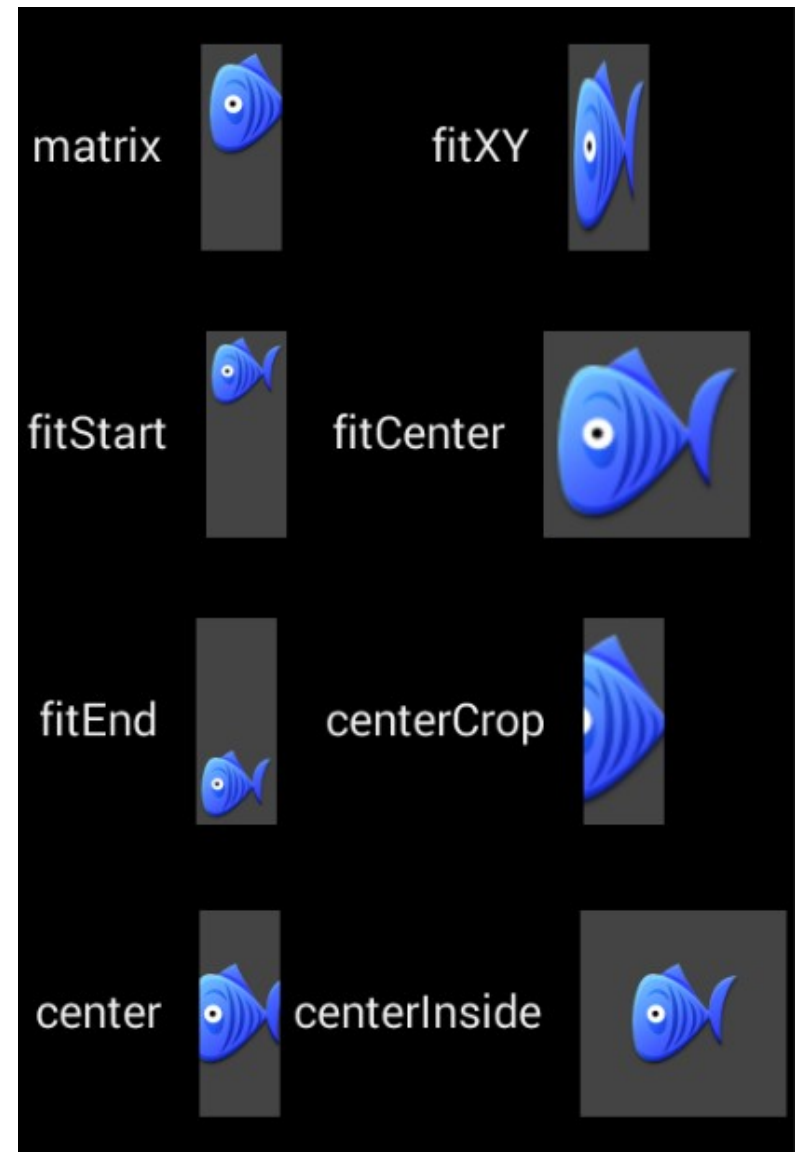


- Affiche une image sans interaction (cf. ImageButton)
- Propriétés importantes :
 - ➔ `android:src` : l'id du drawable
 - ➔ `android:scaleType` : contrôle comment l'image va être modifiée en fonction de la taille de la vue



ImageView - ScaleType

- **matrix** : en haut à gauche, coupe si nécessaire
- **fitXY** : étire/comprime l'image en X et Y
- **fitStart**, **fitCenter**, **fitEnd** : réduit/agrandit en conservant les proportions
- **center** : conserve la taille, coupée si nécessaire et centre
- **centerCrop** : étire/comprime l'image. Coupe l'image si la proportion n'est pas respectée
- **centerInside** : centre et réduit si nécessaire (n'agrandit pas <> fitCenter)



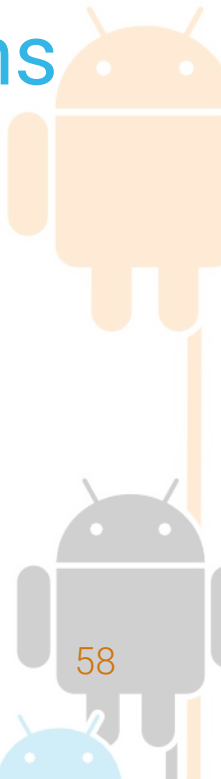
IN01 – Séance 03

Layouts

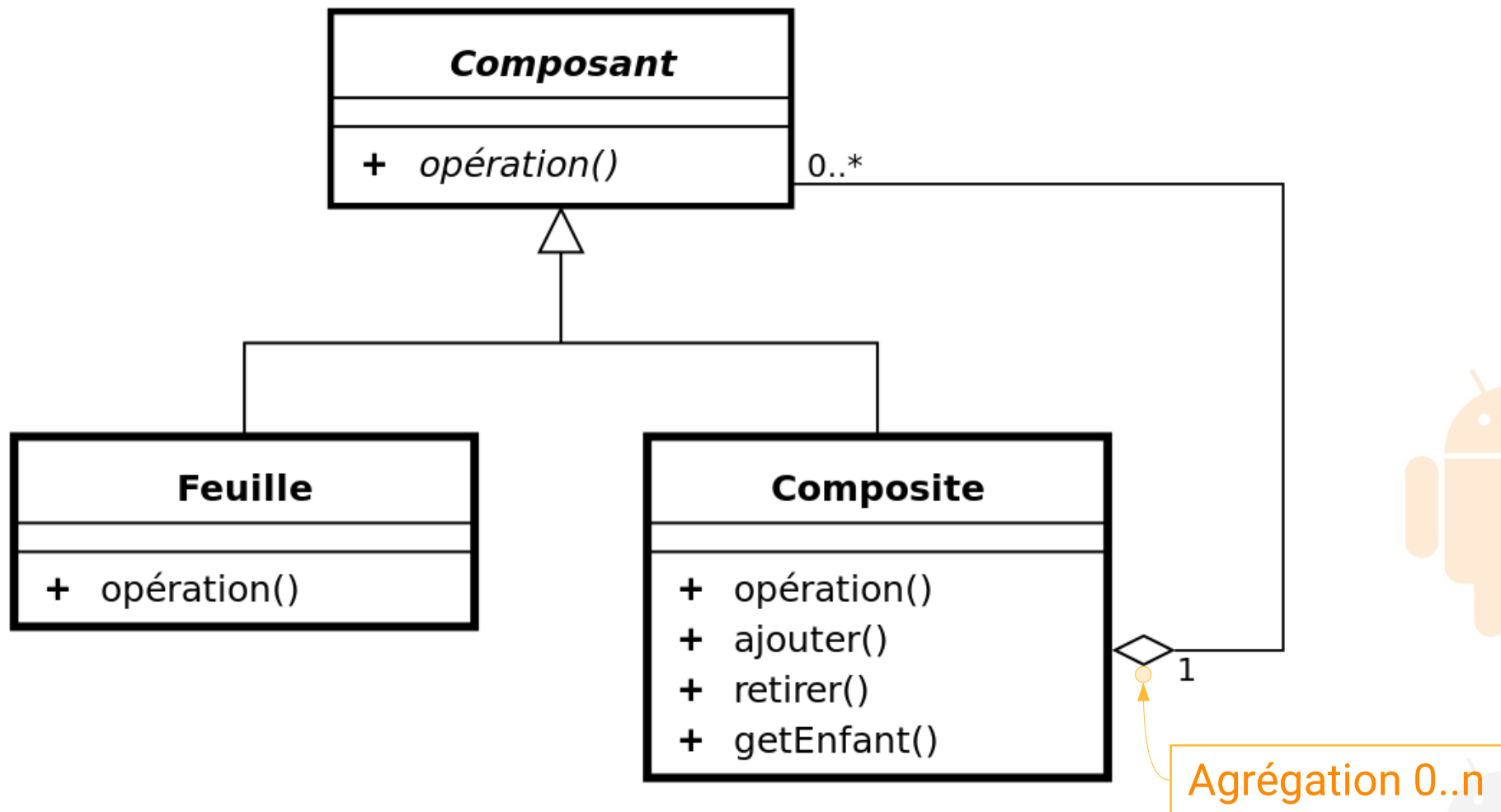


Définition

- Des conteneurs capables de placer les composants qu'ils contiennent de façon automatique
- Plusieurs conteneurs == plusieurs comportements différents == plusieurs façons de placer les composants sur la vue.



Patron - Composite



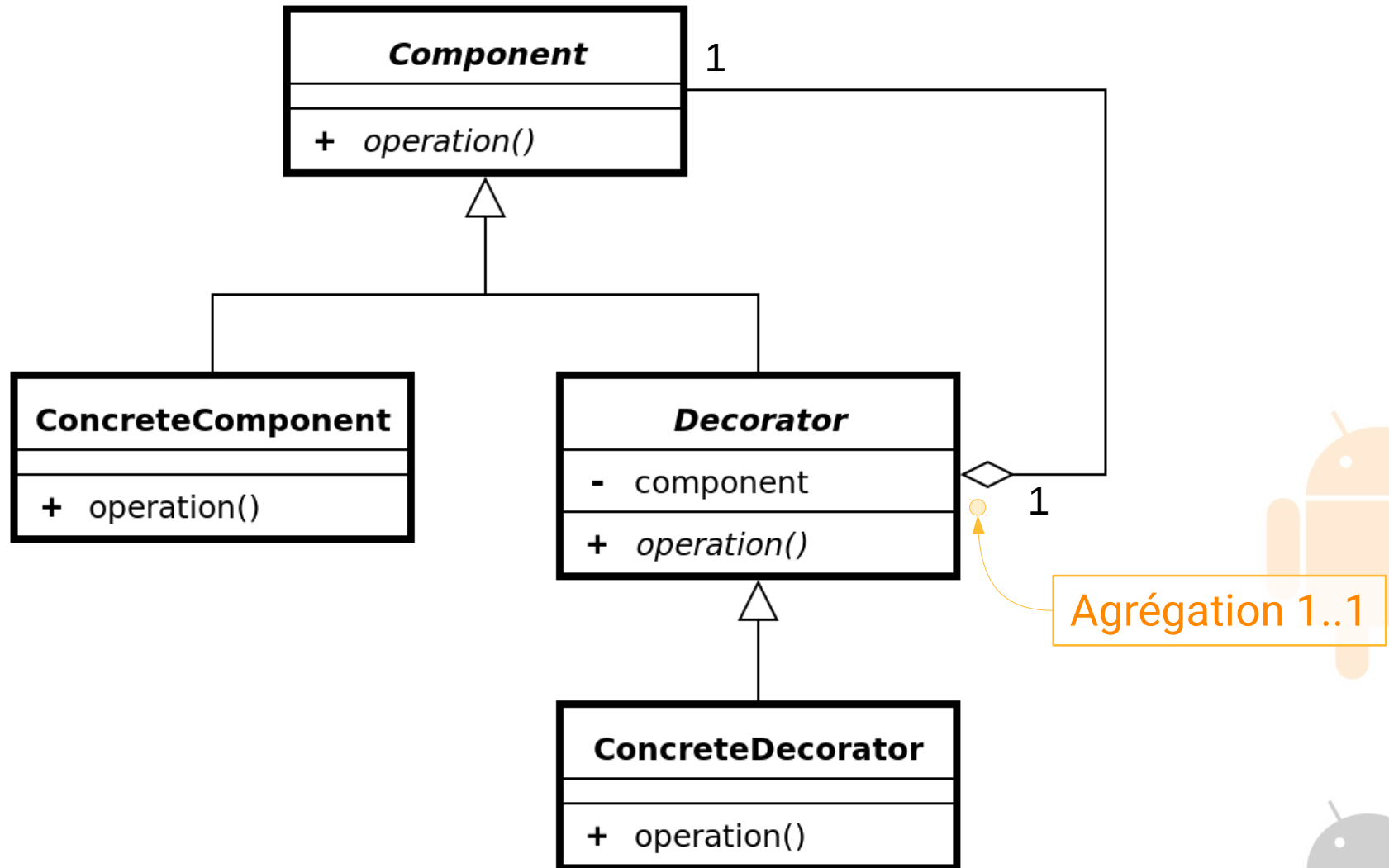
Patron - Composite

- Motivation : manipuler un groupe d'objets/une hiérarchie
- Un composant peut être :
 - ➔ Une feuille (un item : un composant unique)
 - ➔ Un Composite (un node : un composant composé d'autres composants)

```
public abstract class Component {  
    public abstract void doStuff();  
}  
  
public class Composite extends Component{  
  
    private final List<Component> children = new ArrayList<Component>();  
    public void addChild(Component child) {  
        children.add(child);  
    }  
  
    @Override public void doStuff() {}  
}  
  
public class Leaf extends Component {  
    @Override public void doStuff() {}  
}
```

Agrégation 1..*

Patron - Decorator



Patron - Decorator

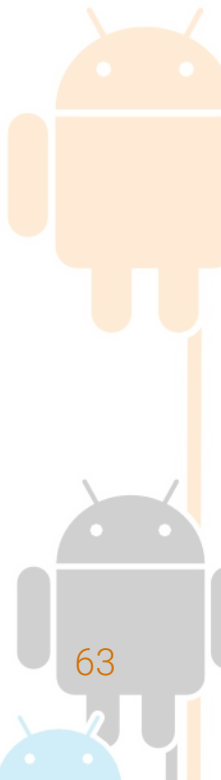
- Motivation : Ajouter dynamiquement une nouvelle responsabilité à un objet
- Un composant peut être :
 - Une feuille (un item : un composant unique)
 - Un décorateur (un composant qui encapsule un et un seul autre)
- Le décoré est généralement passé dans le constructeur (immutable)

```
public class Decorator extends Component {  
    private Component decored;  
  
    public Decorator(Component decored) {  
        this.decored = decored;  
    }  
  
    @Override public void doStuff() {  
        // alter before  
        decored.doStuff();  
        // alter after  
    }  
}
```

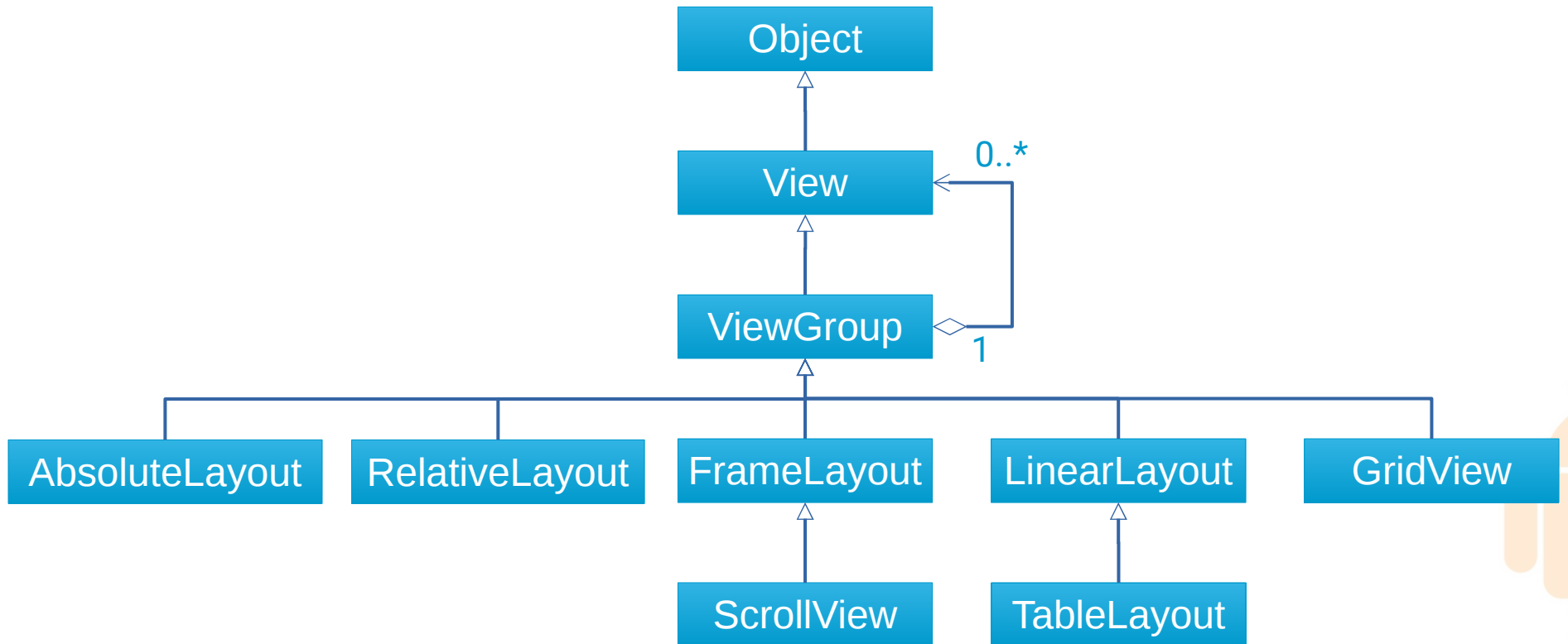
Altère le comportement

Résumé

- Le patron composite, permet de créer des vues composées : les layouts en général (qui héritent de l'objet ViewGroup)
- Le patron decorator, permet d'altérer une vue : ScrollView
 - ➔ C'est un composite déguisé, qu'un seul enfant

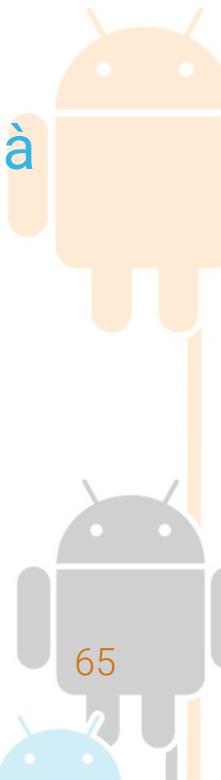


Relation d'héritage



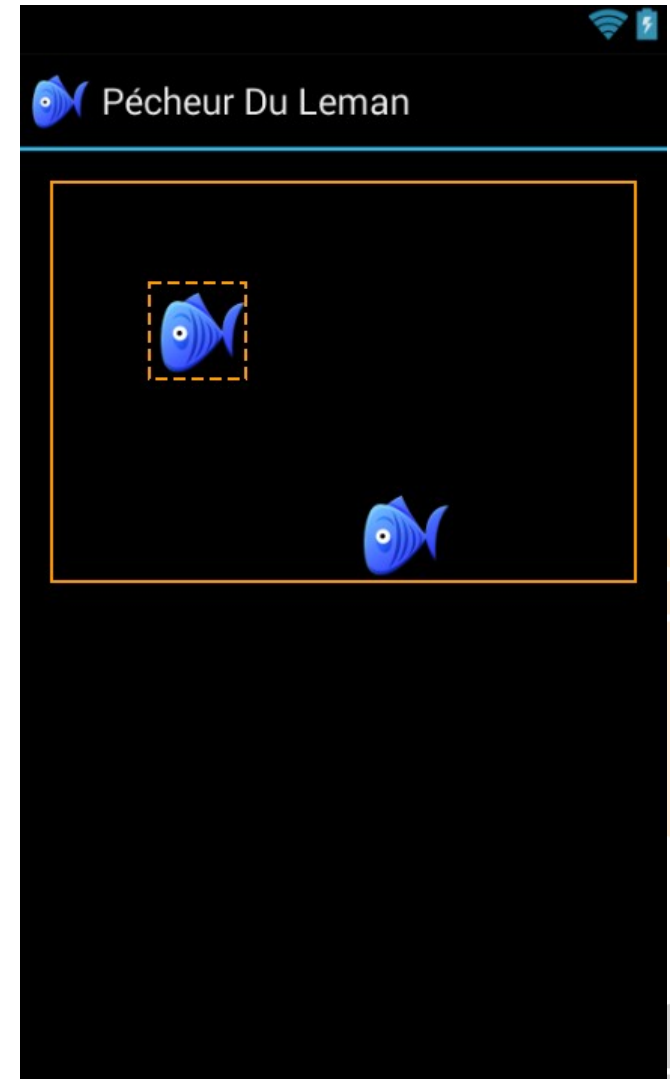
Vue d'ensemble

- `AbsoluteLayout` : position absolue (x et y)
- `LinearLayout` : les composants, les uns au-dessus des autres (ou à côté, selon l'orientation)
- `RelativeLayout` : la position des composants est relative aux autres
- `TableLayout` : crée un tableau à deux dimensions organisé par colonne et par rangée
- `GridLayout` : grille flexible, liste d'éléments continus (de gauche à droite)
- `FrameLayout` : comme des “calques”, les composants les uns derrière les autres



AbsoluteLayout

- Placement des composants par positionnement
- Propriétés des composants :
 - ➔ `android:layout_x` et `android:layout_y`



LinearLayout

- Position en colonnes ou en lignes
- Deux orientations possibles grâce à la propriété `android:layout_orientation`
 - `vertical` : positionne les vues les unes au-dessus des autres
 - `horizontal` : positionne les vues les unes à côté des autres

The screenshot shows an Android application interface titled "Pêcheur Du Lemman". It features a vertical `LinearLayout` containing five text input fields, each preceded by a label: "Bait name:", "Brand:", "Price (\$):", "Size (in):", and "Weight (oz):". At the bottom of the form, there are two buttons labeled "Save" and "Cancel". The interface is overlaid with a blue grid and handles, indicating it is in a design or development tool.

LinearLayout - Remplissage

- Propriétés Android :
 - ➔ `android:layout_width` : comportement en largeur du composant
 - ➔ `android:layout_height` : comportement en hauteur
- Valeurs de remplissage (type de comportement) :
 - ➔ `match_parent` : remplit l'emplacement que lui autorise le conteneur
 - ➔ `wrap_content` : si possible, force le parent à respecter la taille du contenu du composant

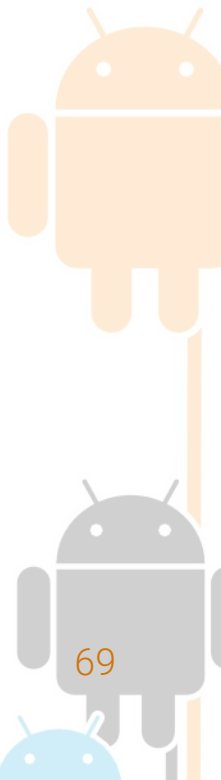
LinearLayout - Poids et Gravity

- Poids :

- ➔ `android:layout_weight` : détermine le poids du composant par rapport aux autres
- ➔ Plus le poids est important, plus le composant prend de la place

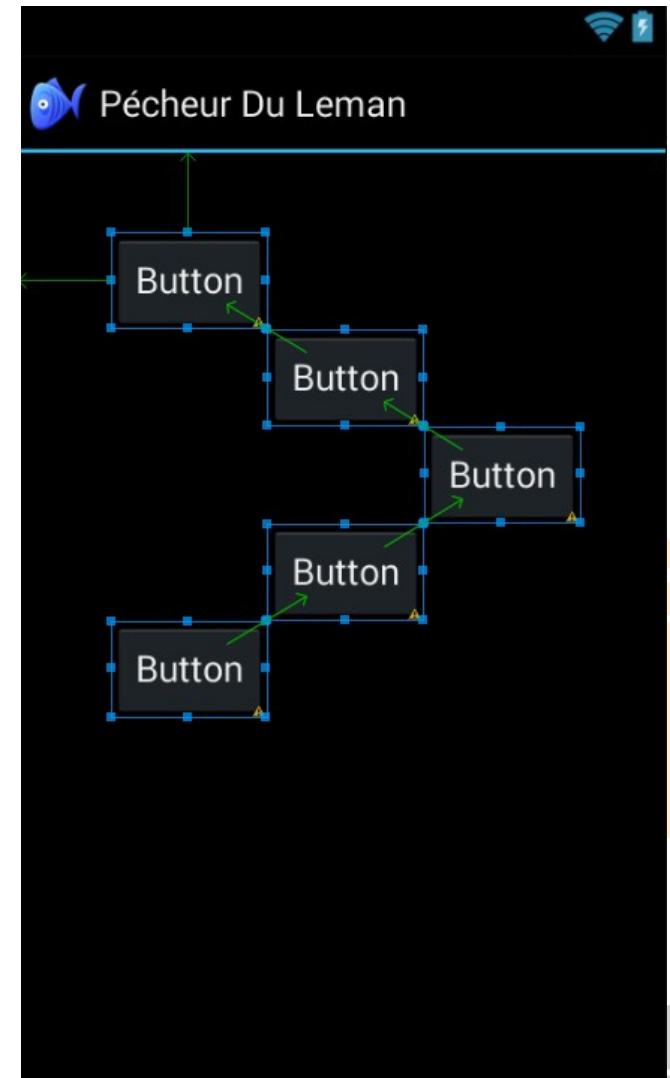
- Gravity :

- ➔ Gestion de l'espace vide : gère l'emplacement du composant dans la case
- ➔ Propriété Android : `android:layout_gravity`



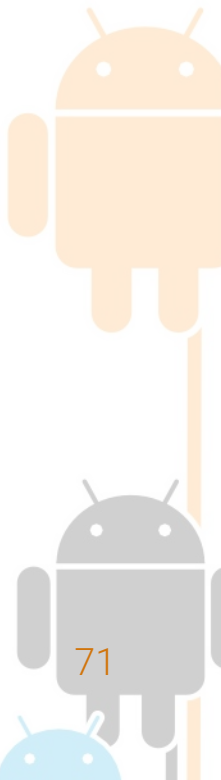
RelativeLayout

- Positionne de façon relative
- Relatif au composant :
 - `android:layout_below` : en dessous de
 - `android:layout_above` : au-dessus de
- Position des bords :
 - `layout_toLeftOf`,
`layout_toRightOf`,
`layout_toTopOf`,
`layout_toBottomOf`



RelativeLayout

- Espacement avec les composants relatifs :
 - ➔ `layout_marginTop`, `layout_marginLeft`,
`layout_marginBottom`,
`layout_marginRight`
- Occupation de la case :
 - ➔ `layout_width` et `layout_height`



RelativeLayout - Exemple

<RelativeLayout>

<Button

```
android:id="@+id/btn1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentLeft="true"
android:layout_alignParentTop="true"
android:layout_marginLeft="30dp"
android:layout_marginTop="23dp"/>
```

<Button

```
android:id="@+id/btn2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@id/btn1"
android:layout_toRightOf="@id/btn1"/>
```

<Button

```
android:id="@+id/btn3"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@id/btn2"
android:layout_toRightOf="@id/btn2"/>
```

marges

<Button

```
android:id="@+id/btn4"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@id/btn3"
android:layout_toLeftOf="@id/btn3"/>
```

<Button

```
android:id="@+id/btn5"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@id/btn4"
android:layout_toLeftOf="@+id/btn4"/>
```

</RelativeLayout>

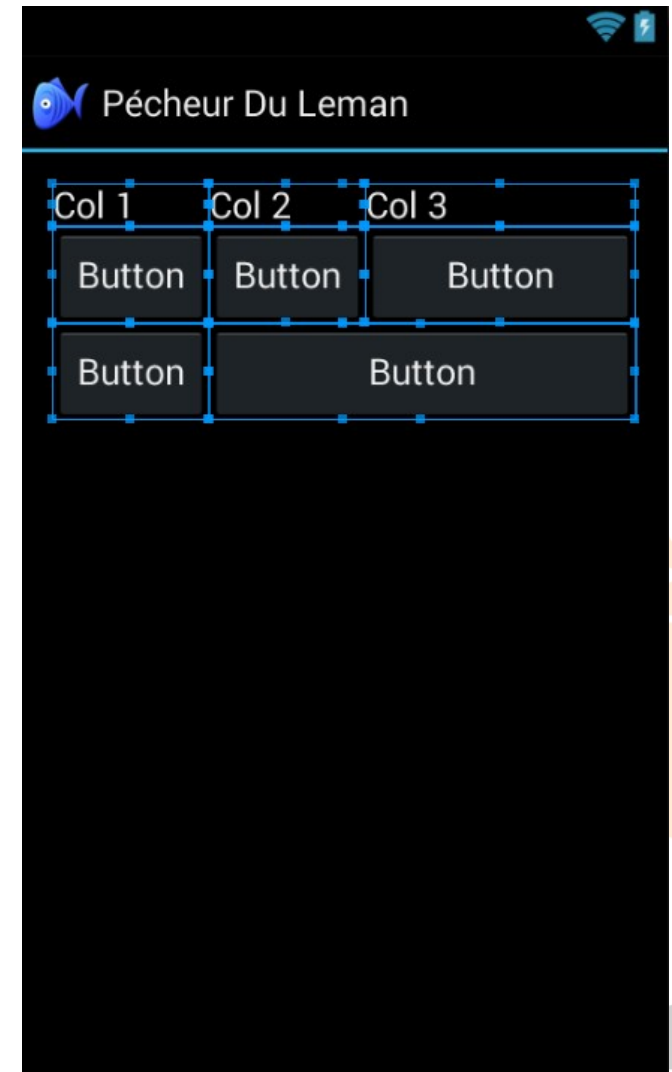
Inutile de créer un nouvel ID

Relatif à et aligné sur

TableLayout

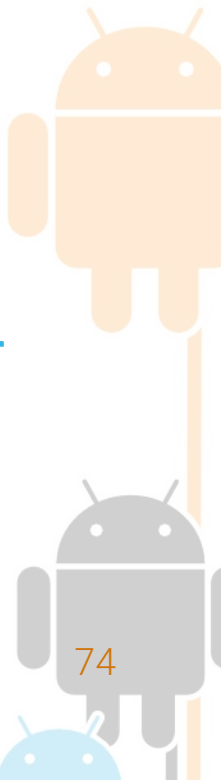
- Un tableau comme en HTML
- Construction par colonnes et rangées

```
<TableLayout>  
  
  <TableRow></TableRow>  
  <TableRow></TableRow>  
  <TableRow></TableRow>  
  
</TableLayout>
```



TableLayout

- Propriétés du tag `<TableLayout>` :
 - ➔ `android:shrinkColumns` : colonne qui sera réduite si l'espace devient trop petit
 - ➔ `android:stretchColumns` : colonne qui sera augmentée
- Propriétés du tag `<TableRow>` :
 - ➔ `android:paddingTop` : espacement haut interne
- Propriétés des composants enfants :
 - ➔ `android:layout_span` : occupation du composant sur plusieurs colonnes



TableLayout - Exemple

```
<TableLayout
  android:shrinkColumns="2"
  android:stretchColumns="2">

  <TableRow
    android:id="@+id/tableRow1"
    android:paddingTop="50dp" >

    <TextView android:id="@+id/textView1" />
    <TextView android:id="@+id/textView2" />
    <TextView android:id="@+id/textView3" />

  </TableRow>

  <TableRow android:id="@+id/tableRow2" >

    <Button android:id="@+id/button1" />
    <Button
      android:id="@+id/button2"
      android:layout_span="2" />

  </TableRow>
</TableLayout>
```

Adaptation selon la taille

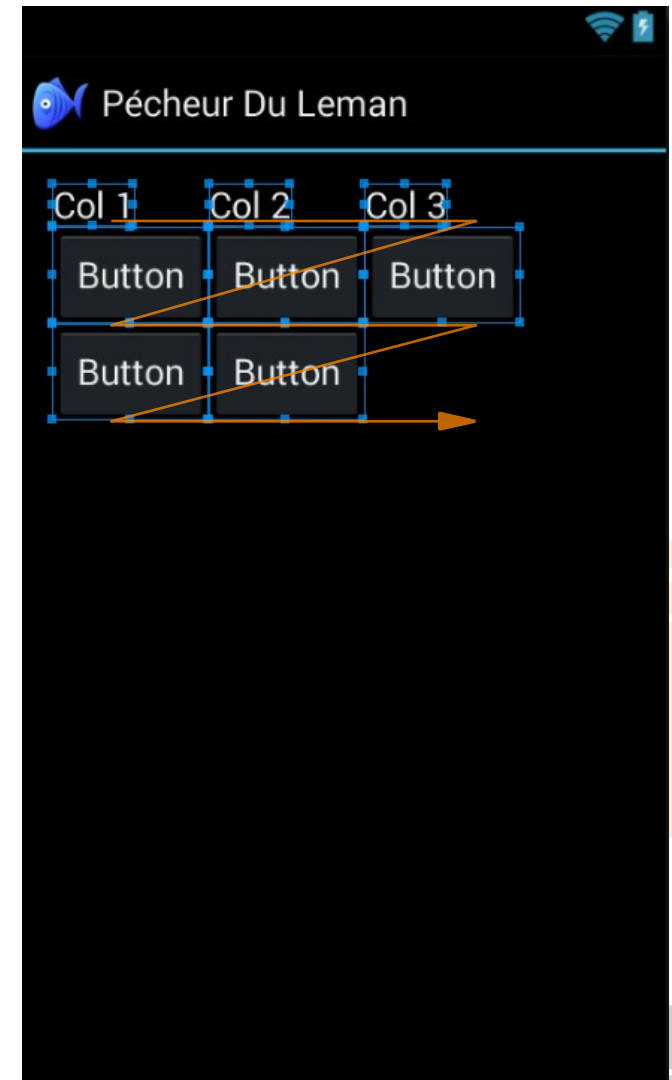
Rangée

Marge intérieure

Occupation

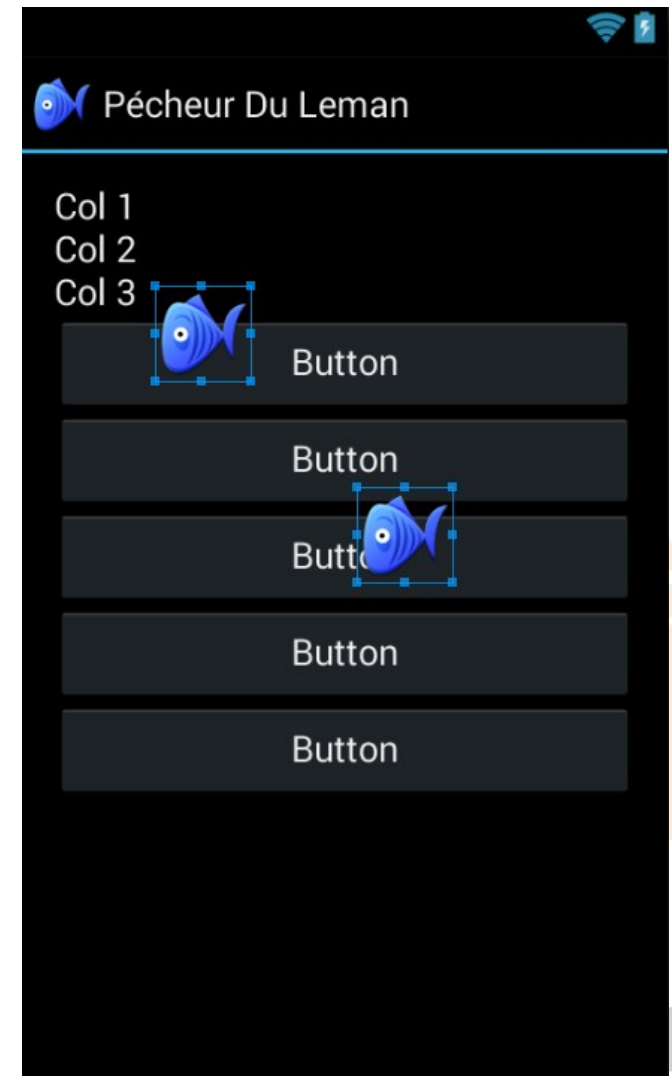
GridLayout

- Remplit la grille de gauche à droite ou de haut en bas
- Deux orientations possibles : **horizontal** et **vertical**
- Propriétés importantes :
 - ➔ `android:columnCount` : nombre de colonnes avant retour à la ligne
 - ➔ `android:rowCount` : nombre de rangées avant retour au début
 - ➔ `android:layout_columnSpan` et `android:layout_rowSpan` pour gérer l'occupation



FrameLayout

- Permet de placer des composants en surimpression
- Des panneaux glissants
- Des menus
- Plus de détails au cours 05
- Ou à l'adresse :
<http://blog.neteril.org/blog/2013/10/10/framelayout-your-best-ui-friend/>



Conclusion

- Framework très bien conçu !
- Plus simple et plus intuitif que Swing
- FrameLayout offre de grandes possibilités
- Ne pas hésiter à imbriquer les layouts entre eux (le composite est notre ami !)
- Approches complémentaires : par WYSIWYG et en XML

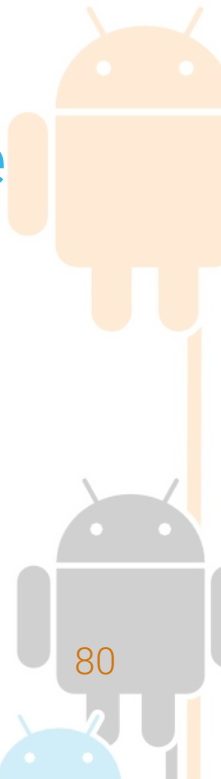
IN01 – Séance 03

Menus



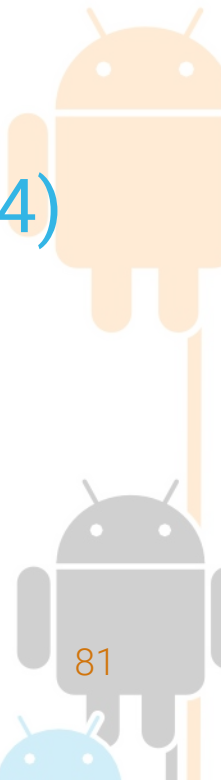
Menus

- Trois types de menus
 - ➔ Option menu : menu standard pour naviguer dans l'application
 - ➔ Popup menu : les actions secondaires (après l'option menu en général)
 - ➔ Menu contextuel : en fonction du contenu touché



Menus

- Accessibles depuis le bouton Menu de l'appareil
- Les menus sont placés différemment selon la version d'Android :
 - ➔ En bas jusqu'à Android 3.2.2 (API ≤ 13)
 - ➔ En haut à droite à partir d'Android 4.0.1 (API ≥ 14)



Déclaration en XML

- Il faut créer un fichier `res/menu/my_menu.xml`

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

    <item
        android:id="@+id/menu_file"
        android:showAsAction="ifRoom|withText"
        android:title="@string/file">
        <menu>
            <item
                android:id="@+id/menu_new"
                android:title="@string/newf"/>
            <item
                android:id="@+id/menu_open"
                android:title="@string/openfile"/>
        </menu>
    </item>
    <item
        android:id="@+id/menu_settings"
        android:title="@string/settings"/>
</menu>
```

Sous-menu

Visible dans la
barre d'action

Dans l'activity

- Les menus peuvent être différents selon les activities
- On utilise la méthode `onCreateOptionsMenu` et l'objet `MenuInflater`
- Les inflaters sont des objets qui créent des vues à partir de fichiers XML

```
public class MainActivity extends Activity {  
  
    @Override  
    public boolean onCreateOptionsMenu(Menu menu) {  
  
        getMenuInflater().inflate(R.menu.main, menu);  
        return true;  
    }  
}
```

Le menu est chargé

Le lien vers le fichier XML

Évènement

- Trois façons d'y parvenir :
 - ➔ Redéfinir `Activity.onOptionsItemSelected (MenuItem item)`
 - Il faudra comparer les id avec un switch sur la méthode `item.getItemId()`
 - ➔ Dans `onCreateOptionsMenu`, pour chaque menuitem, ajouter un évènement `onMenuItemClickListener`
 - ➔ Dans le XML, ajouter la méthode `onClick`. Attention à la signature de la méthode, elle prend comme paramètre un `MenuItem` et non une `View`



onOptionsItemSelected

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
  
    int id = item.getItemId();  
  
    switch (id) {  
        case (R.id.action_example):  
            // menu example  
            return true;  
        case (R.id.action_settings):  
            // menu settings  
            return true;  
        default:  
            return super.onOptionsItemSelected(item);  
    }  
}
```

Le menu est chargé

Cascade l'appel avec la super class

onOptionsItemSelected

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);

    MenuItem item1 = menu.findItem(R.id.action_example);
    item1.setOnMenuItemClickListener(new OnMenuItemClickListener() {
        @Override
        public boolean onMenuItemClick(MenuItem item) {
            // menu example
            return true;
        }
    });

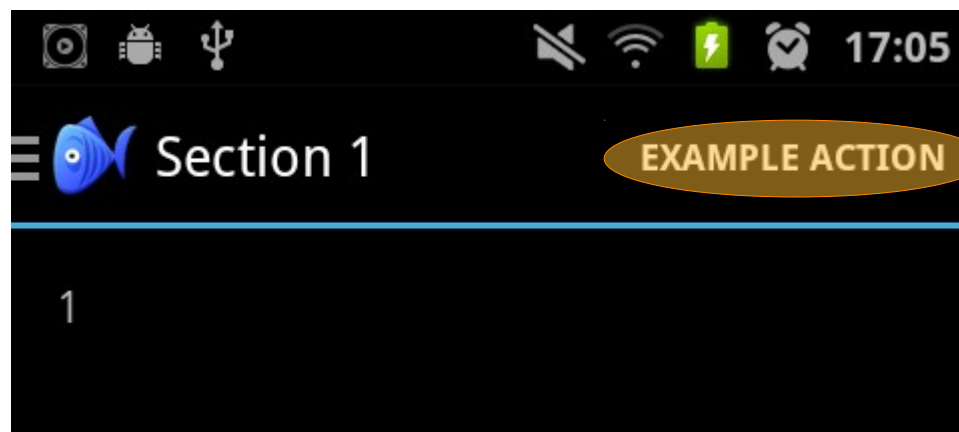
    MenuItem item2 = menu.findItem(R.id.action_settings);
    item2.setOnMenuItemClickListener(new OnMenuItemClickListener() {
        @Override
        public boolean onMenuItemClick(MenuItem item) {
            // menu settings
            return true;
        }
    });

    return true;
}
```

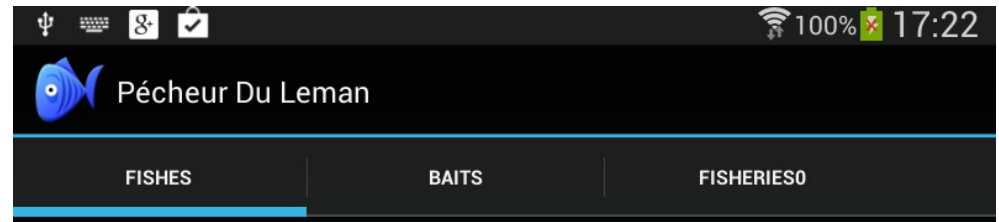
Un évènement pour chaque MenuItem

ActionBar

- Depuis la version 11 de l'API (Android 3.0 HoneyComb) il est possible de sortir certains MenuItem pour un accès direct dans la barre d'action. Il faut utiliser le paramètre `android:showAsAction`



Onglets



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

    ActionBar actionBar = getActionBar();
    actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

    ActionBar.Tab tab1 = actionBar.newTab();
    tab1.setText("Fishes");
    tab1.setTabListener(new TabListener() {
        @Override
        public void onTabUnselected(Tab tab, FragmentTransaction ft) {
            // Do some stuff
        }

        @Override
        public void onTabSelected(Tab tab, FragmentTransaction ft) {
            // Do some stuff
        }

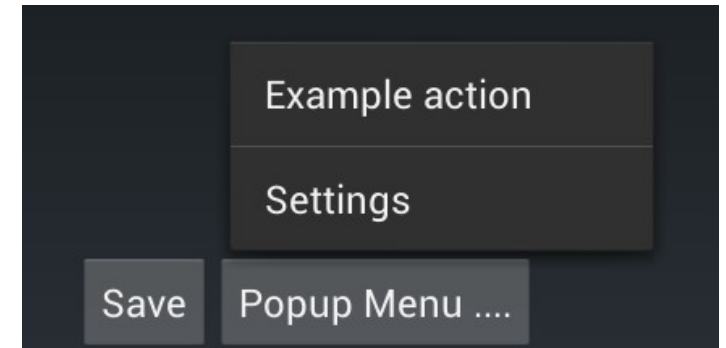
        @Override
        public void onTabReselected(Tab tab, FragmentTransaction ft) {
            // Do some stuff
        }
    });
    actionBar.addTab(tab1);
    // etc.
}
```

Callback obligatoire

- API >= 11 ou appCompatv7
- On peut créer une navigation avec des onglets (NAVIGATION_MODE_TABS)

PopupMenu

- API >= 11 ou appCompatv7



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

    final Button bt = (Button)findViewById(R.id.buttonMenu);
    bt.setText("Popup Menu ....");
    bt.setOnClickListener(new OnClickListener() {

        @Override
        public void onClick(View v) {
            PopupMenu menu = new PopupMenu(MainActivity2.this, bt);
            MenuInflater inflater = menu.getMenuInflater();
            inflater.inflate(R.menu.main, menu.getMenu());
            menu.show();
        }
    });
}
```

Création du menu popup

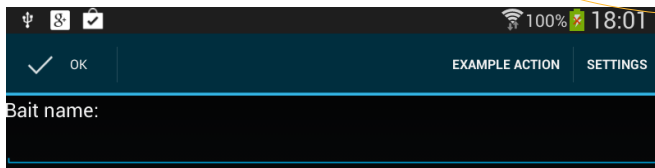
Menu Contextuel

Création du menu

Évènements clic

Démarre le menu

```
callback = new Callback() {  
  
    @Override  
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {  
        // création  
        MenuInflater inflater = MainActivity2.this.getMenuInflater();  
        inflater.inflate(R.menu.main, menu);  
        return true;  
    }  
  
    @Override  
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {  
        // appelé juste avant la création du menu  
        return false;  
    }  
  
    @Override  
    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {  
        // Récupérer les évènements  
        return false;  
    }  
  
    @Override  
    public void onDestroyActionMode(ActionMode mode) {  
        // appelé à la destruction de celui-ci  
    }  
};  
  
TextView bait = (TextView) findViewById(R.id.textBait);  
bait.setOnLongClickListener(new OnLongClickListener() {  
  
    @Override  
    public boolean onLongClick(View v) {  
  
        actionMode = MainActivity2.this.startActionMode(callback);  
        return true;  
    }  
});
```



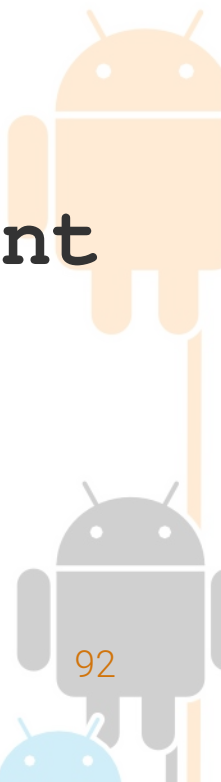
IN01 – Séance 03

Intents et chaînage des activities



Intent

- Une intent (intention) est une description abstraite d'une action à effectuer par l'appareil
- Message asynchrone interprocessus
- Interagir avec des composants internes et externes à l'application
- Classe abstraite : **`android.content.Intent`**



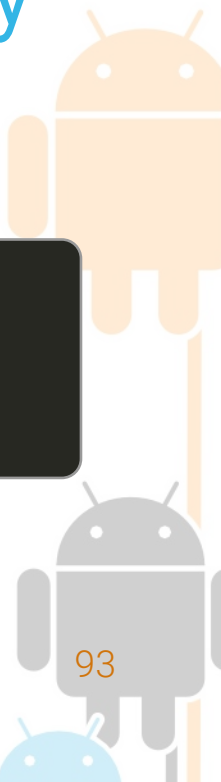
Chaînage des écrans

- Pour passer d'un écran à un autre, on utilise une intention :

```
Intent intent = new Intent(this, MainActivity2.class);
startActivity(intent);
```

- On n'oublie pas de déclarer la nouvelle activity dans le manifest

```
<activity
    android:name="fr.cnam.pecheurduleman.MainActivity2"
    android:label="@string/global_app_name">
</activity>
```



Passage de données entre activities

- Les intents permettent de passer des informations grâce à la méthode `putExtra()`

```
Intent i = new Intent(this, MainActivity2.class);  
i.putExtra("myExtra1", "myStringValue");  
i.putExtra("myExtra2", 10);  
startActivity(i);
```

- Que l'on récupère dans la nouvelle activity :

```
String myExtra1 = getIntent().getExtras().getString("myExtra1", "default");  
int myExtra2 = getIntent().getExtras().getInt("myExtra2", 0);
```

Valeur par défaut

IN01 – Séance 03

Toast



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

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

- Attention construire le Toast ne l'affiche pas : il faut utiliser **show()** pour cela

Fin

- Merci de votre attention
- Des questions ?

