

Les interfaces graphiques



Sources : <http://www.android.com>, wikipedia...

le cnam

Paris, 28/11/2017

Interface Graphique

2

- Pour définir l'interface graphique d'une application dans une activité, Android propose un *framework* très riche : une structure hiérarchique faite de vues et de conteneurs.
- 2 approches pour spécifier l'interface graphique :
 - une approche programmatique «classique» en *Java*.
 - une approche basée sur une description *XML*
- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches.
- Interaction à travers des *EventListener* et par redéfinition de «*callbacks*».

Spécification de l'IHM en XML

3

- Une analogie très grande avec le modèle du Web :
 - Des *layout XML* (fichiers ressources de type *layout*) jouent le rôle du *HTML* (structure arborescente de la présentation des données).
 - Des styles *XML* (fichiers ressources spécifiques) jouent le rôle des feuilles de style *CSS*.
 - Les programmes *Java* jouent le rôle des codes *JavaScript*
 - Le système Android joue le rôle du navigateur
- Un activité demande au système de charger un *layout*. Le système crée alors une arborescence d'objets *Java* à partir de cette description (équivalent au *DOM*)

Spécification de l'IHM en XML

4

- On récupère la référence d'un objet créé en *XML* par la méthode `findViewById()` de la classe `Activity`.
- Pour maintenir la correspondance entre le «monde *Java*» et le «monde *XML*» le processus de compilation crée une classe particulière : la classe `R`. La classe `R` est mise à jour automatiquement par le SDK à chaque compilation en cas de modification des fichiers ressources (du contenu du dossier `res/`).
- Les interactions peuvent être également spécifiées à partir des fichiers *layout* au moyen d'attributs particuliers dans le *XML* : `android:onClick`, `android:clickable`, `android:longClickable`, `android:focusable`...

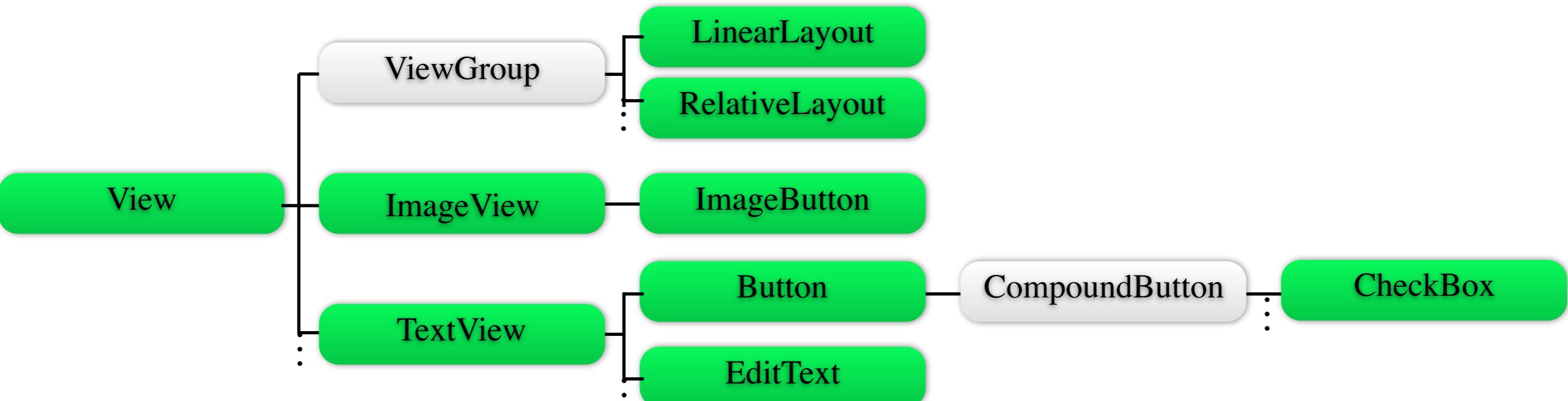
Les widgets (apparence et interaction)

5

- Le système Android propose un grand nombre de *widgets* adaptés à une interaction sur terminal mobile :
 - De très nombreux widgets sont disponibles :
TextView, EditText, CheckBox, Button,
ToggleButton, RadioGroup, Spinner,
AutoCompleteTextView, DatePicker, TimePicker,
ImageView, ImageButton, SeekBar, RatttingBar,
Gallery...
 - Idem pour les *layouts* offrant ainsi une adaptation automatique à différents types d'aspect ratio de l'écran : ListView, FrameLayout, LinearLayout,
TableLayout, RelativeLayout, GridLayout...

Superposition de 2 structures arborescentes

- Structure hiérarchique de classes :



- Structure hiérarchique spatiale de conteneurs :

Superposition de 2 structures arborescentes

The screenshot shows the Android Developers website at <https://developer.android.com/reference/android/widget/TextView.html>. The page title is "TextView". The left sidebar shows navigation links for Reference, droid APIs, API level: 19, and various package names like android.service.voice, android.service.vr, etc. The main content area displays the Java code for TextView, its inheritance path from java.lang.Object through android.view.View and android.widget.TextView, and lists of known direct and indirect subclasses. Below the code, a description states that TextView displays text and optionally allows editing, with a note about setting android:textIsSelectable to true. A search bar and developer console links are also visible at the top.

TextView

```
public class TextView
extends View implements ViewTreeObserver.OnPreDrawListener
    < java.lang.Object
        < android.view.View
            < android.widget.TextView
```

Known Direct Subclasses

AppCompatTextView, Button, CheckedTextView, Chronometer, DigitalClock, EditText, RowLayout, TextClock

Known Indirect Subclasses

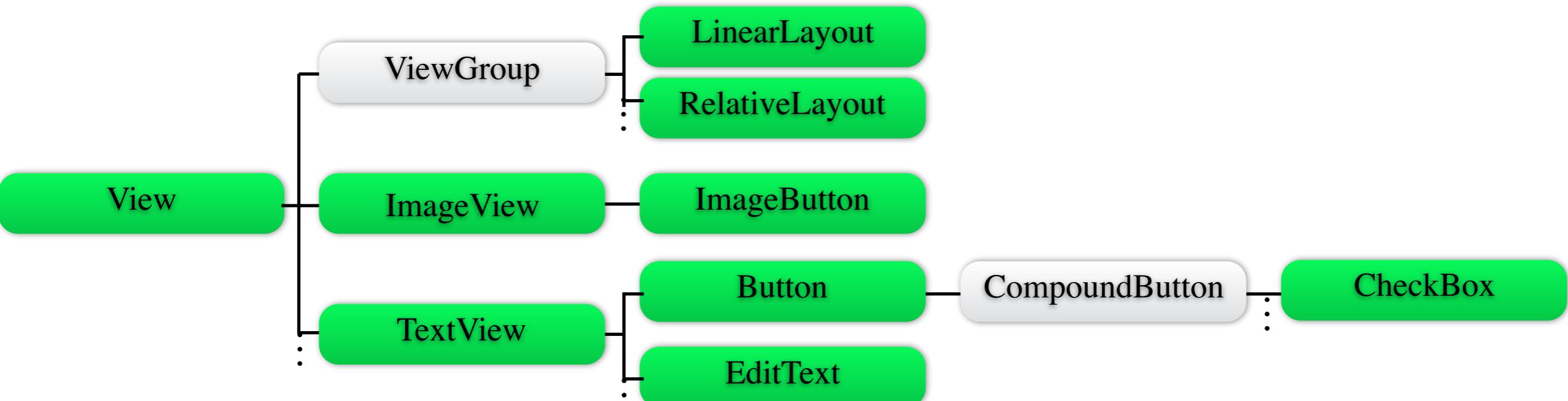
AppCompatAutoCompleteTextView, AppCompatButton, AppCompatCheckBox, AppCompatCheckedTextView, AppCompatEditText, AppCompatMultiAutoCompleteTextView, AppCompatRadioButton, AutoCompleteTextView, CheckBox, CompoundButton, ExtractEditText, GuidedActionEditText, MultiAutoCompleteTextView, RadioButton, SearchEditText, Switch, SwitchCompat, TextInputEditText, ToggleButton

Displays text to the user and optionally allows them to edit it. A TextView is a complete text editor, however the basic class is configured to not allow editing; see [EditText](#) for a subclass that configures the text view for editing.

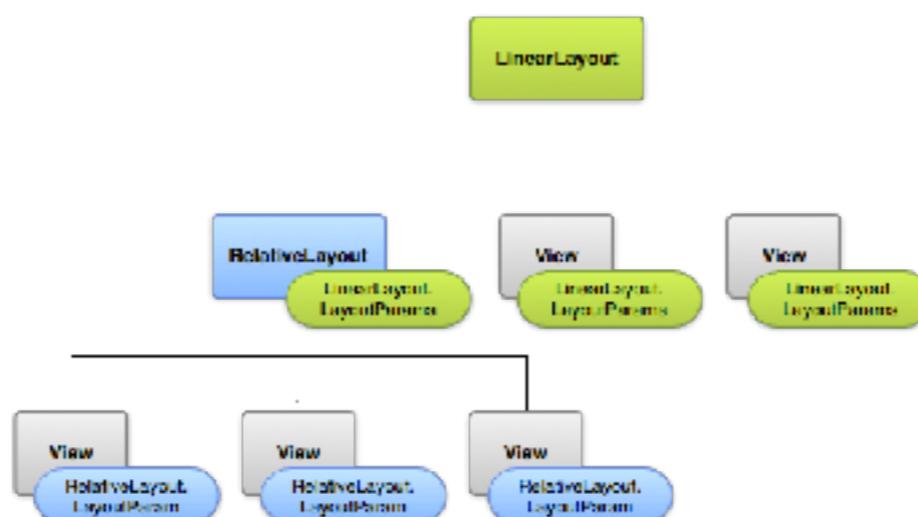
To allow users to copy some or all of the TextView's value and paste it somewhere else, set the XML attribute `android:textIsSelectable` to "true" or call `setTextIsSelectable(true)`. The `textIsSelectable` flag allows users to make selection gestures in the TextView, which in turn triggers the system's built-in copy/paste controls.

Superposition de 2 structures arborescentes

- Structure hiérarchique de classes :

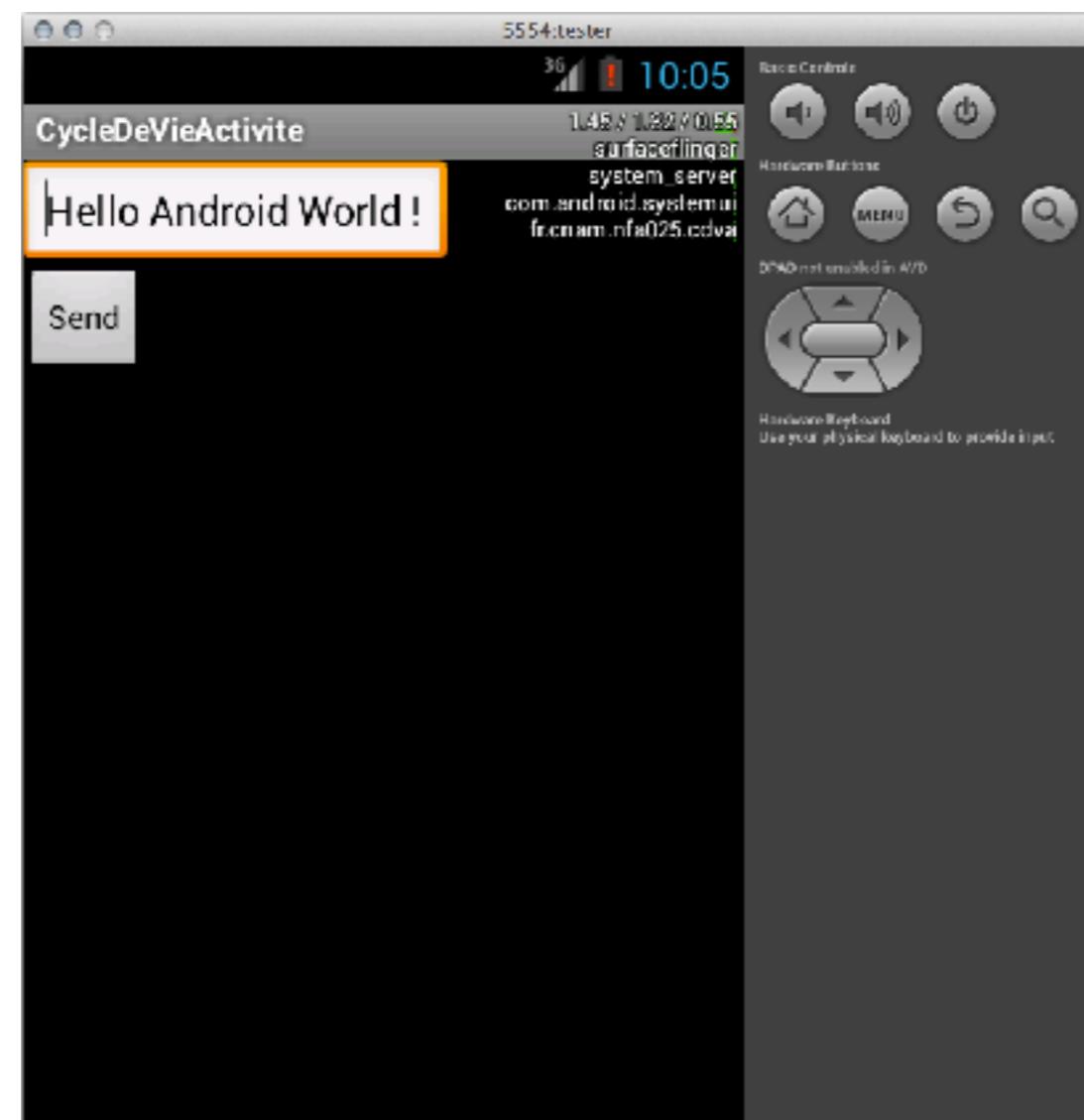


- Structure hiérarchique spatiale de conteneurs :



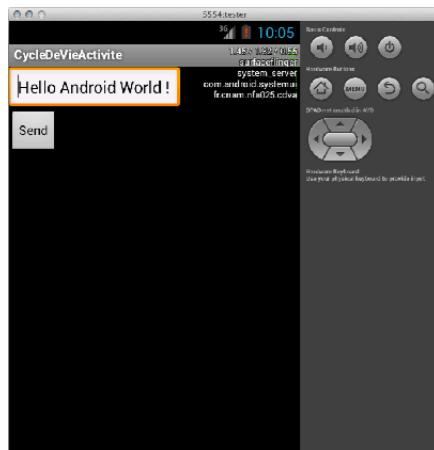
Interface Graphique

- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches



Interface Graphique

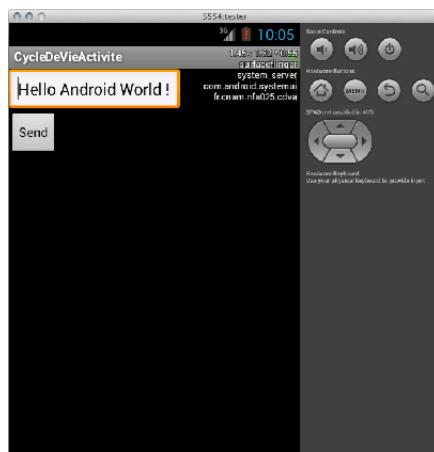
- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:orientation="vertical"
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent"
6     >
7     <EditText
8         android:layout_width="wrap_content"
9         android:layout_height="wrap_content"
10        android:id="@+id/main_msg_te"
11        android:text="@string/LaunchedMessage"
12        />
13     <Button
14         android:layout_width="wrap_content"
15         android:layout_height="wrap_content"
16         android:onClick="actionSendBtn"
17         android:text="@string/send_btn"
18         />
19 </LinearLayout>
```

Interface Graphique

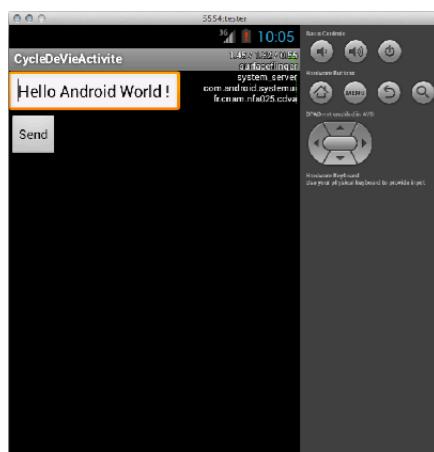
- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:orientation="vertical"
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent"
6     >
7     <EditText
8         android:layout_width="wrap_content"
9         android:layout_height="wrap_content"
10        android:id="@+id/main_msg_te"
11        android:text="@string/LaunchedMessage"
12        />
13     <Button
14         android:layout_width="wrap_content"
15         android:layout_height="wrap_content"
16         android:onClick="actionSendBtn"
17         android:text="@string/send_btn"
18         />
19 </LinearLayout>
```

Interface Graphique

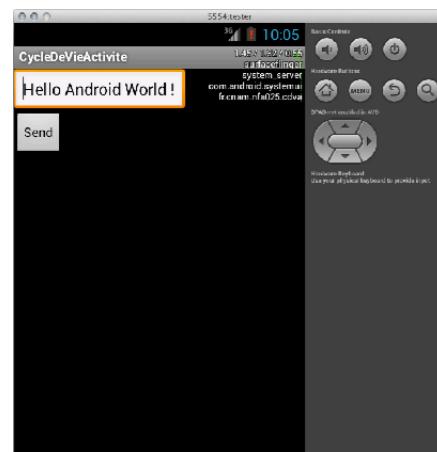
- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:orientation="vertical"
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent"
6     >
7     <EditText
8         android:layout_width="wrap_content"
9         android:layout_height="wrap_content"
10        android:id="@+id/main_msg_te"
11        android:text="@string/LaunchedMessage"
12    />
13    <Button
14        android:layout_width="wrap_content"
15        android:layout_height="wrap_content"
16        android:onClick="actionSendBtn"
17        android:text="@string/send_btn"
18    />
19 </LinearLayout>
```

Interface Graphique

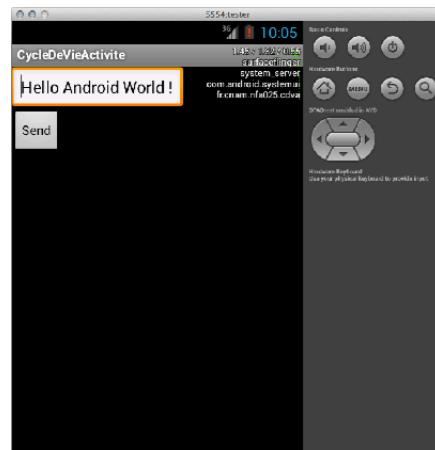
- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches



```
1 /* AUTO-GENERATED FILE.  DO NOT MODIFY.
2 *
3 * This class was automatically generated by the
4 * aapt tool from the resource data it found.  It
5 * should not be modified by hand.
6 */
7
8 package fr.cnam.nfa025.cdva;
9
10 public final class R {
11     public static final class attr {
12     }
13     public static final class id {
14         public static final int main_msg_te=0x7f040000;
15     }
16     public static final class layout {
17         public static final int main=0x7f020000;
18     }
19     public static final class string {
20         public static final int LaunchedMessage=0x7f030002;
21         public static final int app_name=0x7f030000;
22         public static final int send_btn=0x7f030001;
23     }
24 }
```

Interface Graphique

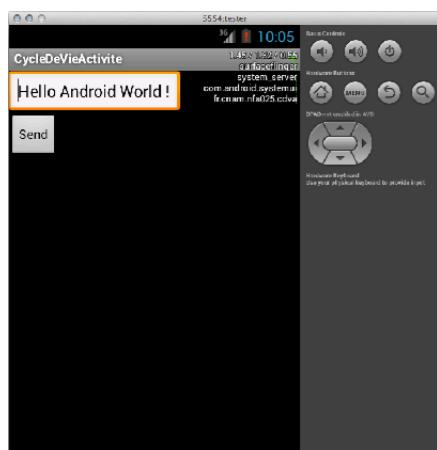
- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches



```
1 /* AUTO-GENERATED FILE.  DO NOT MODIFY.
2 *
3 * This class was automatically generated by the
4 * aapt tool from the resource data it found.  It
5 * should not be modified by hand.
6 */
7
8 package fr.cnam.nfa025.cdva;
9
10 public final class R {
11     public static final class attr {
12     }
13     public static final class id {
14         public static final int main_msg_te=0x7f040000;
15     }
16     public static final class layout {
17         public static final int main=0x7f020000;
18     }
19     public static final class string {
20         public static final int LaunchedMessage=0x7f030002;
21         public static final int app_name=0x7f030000;
22         public static final int send_btn=0x7f030001;
23     }
24 }
```

Interface Graphique

- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches

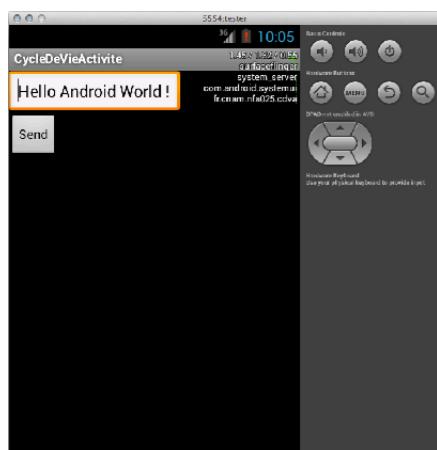


```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3   android:orientation="vertical"
4   android:layout_width="fill_parent"
5   android:layout_height="fill_parent"
6   >
7   <EditText
8     android:layout_width="wrap_content"
9     android:layout_height="wrap_content"
10    android:id="@+id/main_msg_te"
11    android:text="@string/LaunchedMessage"
12  />
13  <Button
14    android:layout_width="wrap_content"
15    android:layout_height="wrap_content"
16    android:onClick="actionSendBtn"
17    android:text="@string/send_btn"
18  />
19 </LinearLayout>
```

```
1 /* AUTO-GENERATED FILE. DO NOT MODIFY.
2 *
3 * This class was automatically generated by the
4 * aapt tool from the resource data it found. It
5 * should not be modified by hand.
6 */
7
8 package fr.cnam.nfa025.cdva;
9
10 public final class R {
11   public static final class attr {
12   }
13   public static final class id {
14     public static final int main_msg_te=0x7f040000;
15   }
16   public static final class layout {
17     public static final int main=0x7f020000;
18   }
19   public static final class string {
20     public static final int LaunchedMessage=0x7f030002;
21     public static final int app_name=0x7f030000;
22     public static final int send_btn=0x7f030001;
23   }
24 }
```

Interface Graphique

- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches



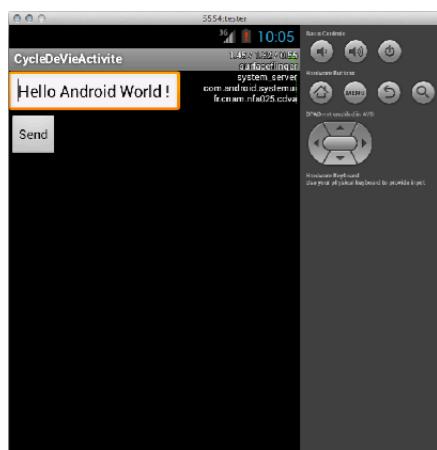
```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3   android:orientation="vertical"
4   android:layout_width="fill_parent"
5   android:layout_height="fill_parent"
6   >
7   <EditText
8     android:layout_width="wrap_content"
9     android:layout_height="wrap_content"
10    android:id="@+id/main_msg_te"
11    android:text="@string/launchedMessage"
12   />
13   <Button
14     android:layout_width="wrap_content"
15     android:layout_height="wrap_content"
16     android:onClick="actionSendBtn"
17     android:text="@string/send_btn"
18   />
19 </LinearLayout>
```

```
1 @Override
2 public void onCreate(Bundle savedInstanceState){
3   super.onCreate(savedInstanceState);
4   this.setContentView(R.layout.main);
5   Log.i(APP_TAG, "CycleDeVieActivite is created!");
6   if(null != savedInstanceState){
7     Log.i(APP_TAG, "CycleDeVieActivite is restoring previous state!");
8     EditText msg_text_widget = (EditText)this.findViewById(R.id.main_msg_te);
9     msg_text_widget.setText(savedInstanceState.getString(BNDL_MSG_KEY));
10  }
11  else{
12    Log.i(APP_TAG, "CycleDeVieActivite : Very first Start");
13  }
14 }
```

```
1 /* AUTO-GENERATED FILE. DO NOT MODIFY.
2 *
3 * This class was automatically generated by the
4 * aapt tool from the resource data it found. It
5 * should not be modified by hand.
6 */
7
8 package fr.cnam.nfa025.cdva;
9
10 public final class R {
11   public static final class attr {
12   }
13   public static final class id {
14     public static final int main_msg_te=0x7f040000;
15   }
16   public static final class layout {
17     public static final int main=0x7f020000;
18   }
19   public static final class string {
20     public static final int LaunchedMessage=0x7f030002;
21     public static final int app_name=0x7f030000;
22     public static final int send_btn=0x7f030001;
23   }
24 }
```

Interface Graphique

- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches



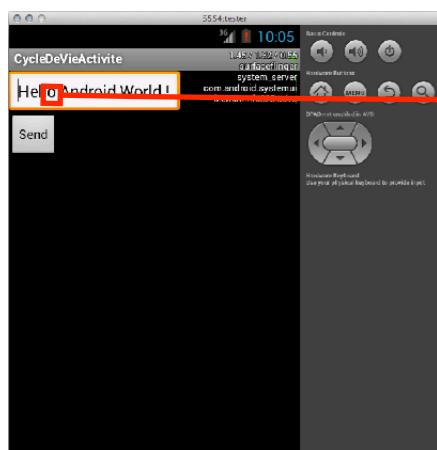
```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3   android:orientation="vertical"
4   android:layout_width="fill_parent"
5   android:layout_height="fill_parent">
6   >
7     <EditText
8       android:layout_width="wrap_content"
9       android:layout_height="wrap_content"
10      android:id="@+id/main_msg_te"
11      android:text="@string/launchedMessage"
12    />
13     <Button
14       android:layout_width="wrap_content"
15       android:layout_height="wrap_content"
16       android:onClick="actionSend"
17       android:text="@string/send_btn"
18     />
19 </LinearLayout>
```

```
1 @Override
2 public void onCreate(Bundle savedInstanceState){
3   super.onCreate(savedInstanceState);
4   this.setContentView(R.layout.main);
5   Log.i(APP_TAG, "CycleDeVieActivite is created!");
6   if(null != savedInstanceState){
7     Log.i(APP_TAG, "CycleDeVieActivite is restoring previous state!");
8     EditText msg_text_widget = (EditText)this.findViewById(R.id.main_msg_te);
9     msg_text_widget.setText(savedInstanceState.getString(BNDL_MSG_KEY));
10   }
11   else{
12     Log.i(APP_TAG, "CycleDeVieActivite : Very first Start");
13   }
14 }
```

```
1 /* AUTO-GENERATED FILE. DO NOT MODIFY.
2 *
3 * This class was automatically generated by the
4 * aapt tool from the resource data it found. It
5 * should not be modified by hand.
6 */
7
8 package fr.cnam.nfa025.cdva;
9
10 public final class R {
11   public static final class attr {
12   }
13   public static final class id {
14     public static final int main_msg_te=0x7f040000;
15   }
16   public static final class layout {
17     public static final int main=0x7f020000;
18   }
19   public static final class string {
20     public static final int LaunchedMessage=0x7f030002;
21     public static final int app_name=0x7f030000;
22     public static final int send_btn=0x7f030001;
23   }
24 }
```

Interface Graphique

- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches



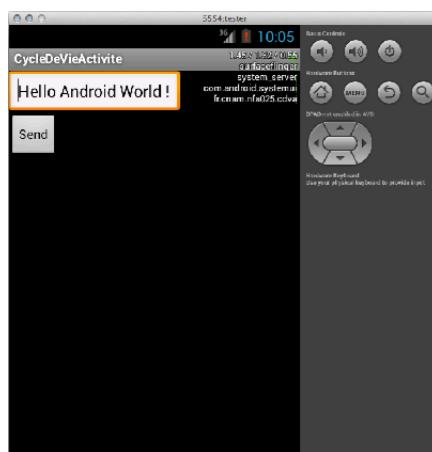
```
1  /* AUTO-GENERATED FILE. DO NOT MODIFY.
2  *
3  * This class was automatically generated by the
4  * aapt tool from the resource data it found. It
5  * should not be modified by hand.
6  */
7
8 package fr.cnam.nfa025.cdva;
9
10 public final class R {
11     public static final class attr {
12     }
13     public static final class id {
14         public static final int main_msg_te=0x7f040000;
15     }
16     public static final class layout {
17         public static final int main=0x7f020000;
18     }
19     public static final class string {
20         public static final int LaunchedMessage=0x7f030002;
21         public static final int app_name=0x7f030000;
22         public static final int send_btn=0x7f030001;
23     }
24 }
```

```
1 @Override
2 public void onCreate(Bundle savedInstanceState){
3     super.onCreate(savedInstanceState);
4     this.setContentView(R.layout.main);
5     Log.i(APP_TAG, "CycleDeVieActivite is created!");
6     if(null != savedInstanceState){
7         Log.i(APP_TAG, "CycleDeVieActivite is restoring previous state!");
8         EditText msg_text_widget = (EditText)this.findViewById(R.id.main_msg_te);
9         msg_text_widget.setText(savedInstanceState.getString(BNDL_MSG_KEY));
10    } else{
11        Log.i(APP_TAG, "CycleDeVieActivite : Very first Start");
12    }
13 }
14
15 <LinearLayout
16     android:layout_width="wrap_content"
17     android:layout_height="wrap_content"
18     android:id="@+id/main_msg_te"
19     android:text="@string/LaunchedMessage"
20     />
21
22 <EditText
23     android:layout_width="wrap_content"
24     android:layout_height="wrap_content"
25     android:id="@+id/send_btn"
26     android:onClick="actionSendBtn"
27     android:text="@string/send_btn"
28     />
29
30 </LinearLayout>
```

Interface Graphique

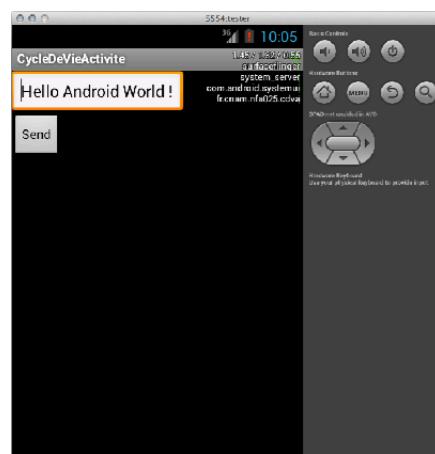
7

- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches



Interface Graphique

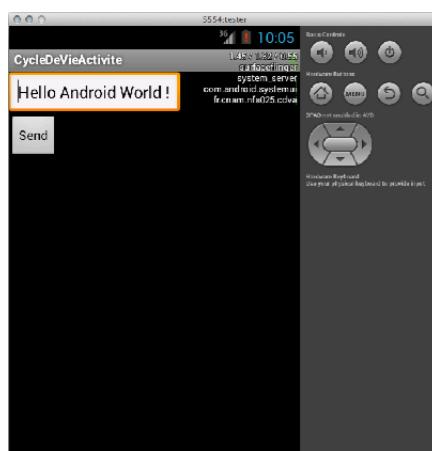
- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches



```
1  @Override
2  public void onCreate(Bundle savedInstanceState){
3      super.onCreate(savedInstanceState);
4      EditText msg_text_widget = new EditText(this);
5      //msg_text_widget.setText(R.string.LaunchedMessage);
6      msg_text_widget.setText("Hello Android World !");
7      Button bt = new Button(this);
8      //bt.setText(R.string.send_btn);
9      bt.setText("Send");
10     LinearLayout lt = new LinearLayout(this);
11     lt.setOrientation(LinearLayout.VERTICAL);
12     lt.addView(msg_text_widget
13             , new LinearLayout.LayoutParams(ViewGroup.LayoutParams.WRAP_CONTENT
14                                         , ViewGroup.LayoutParams.WRAP_CONTENT));
15     lt.addView(bt
16             , new LinearLayout.LayoutParams(ViewGroup.LayoutParams.WRAP_CONTENT
17                                         , ViewGroup.LayoutParams.WRAP_CONTENT));
18     this.setContentView(lt);
19     Log.i(APP_TAG, "CycleDeVieActivite is created!");
20     if(null != savedInstanceState){
21         Log.i(APP_TAG, "CycleDeVieActivite is restoring previous state!");
22         msg_text_widget.setText(savedInstanceState.getString(BNDL_MSG_KEY));
23     }
24     else{
25         Log.i(APP_TAG, "CycleDeVieActivite : Very first Start");
26     }
27 }
```

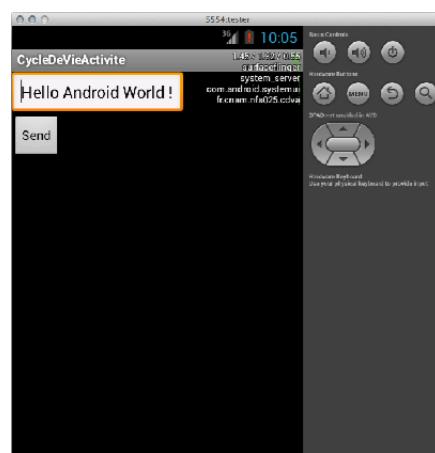
Interface Graphique

- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches
- Interaction à travers des EventListeners ou par redéfinition de «callbacks»



Interface Graphique

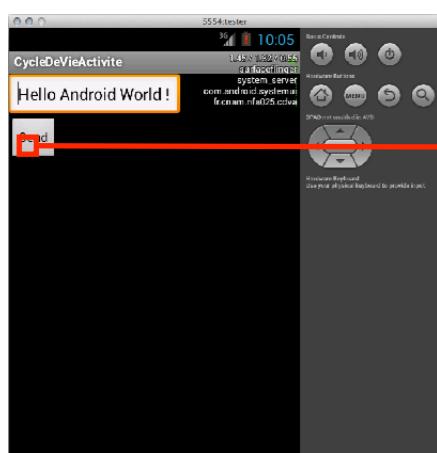
- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches
- Interaction à travers des EventListeners ou par redéfinition de «callbacks»



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:orientation="vertical"
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent"
6     >
7     <EditText
8         android:layout_width="wrap_content"
9         android:layout_height="wrap_content"
10        android:id="@+id/main_msg_te"
11        android:text="@string/LaunchedMessage"
12        />
13     <Button
14         android:layout_width="wrap_content"
15         android:layout_height="wrap_content"
16         android:onClick="actionSendBtn"
17         android:text="@string/send_btn"
18         />
19     </LinearLayout>
```

Interface Graphique

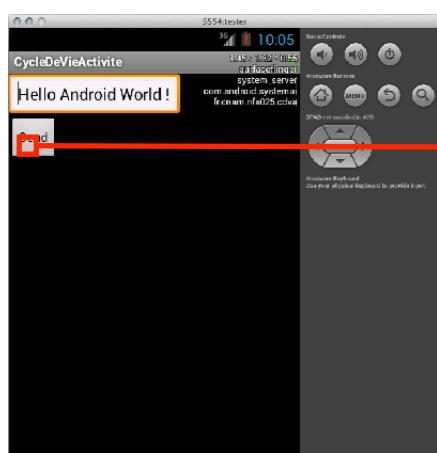
- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches
- Interaction à travers des EventListeners ou par redéfinition de «callbacks»



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:orientation="vertical"
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent">
6
7     <EditText
8         android:layout_width="wrap_content"
9         android:layout_height="wrap_content"
10        android:id="@+id/main_msg_te"
11        android:text="@string/launchedMessage"
12    />
13
14     <Button
15         android:layout_width="wrap_content"
16         android:layout_height="wrap_content"
17         android:id="@+id/main_sendBtn"
18         android:text="@string/send_btn"
19    />
20 </LinearLayout>
```

Interface Graphique

- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches
- Interaction à travers des EventListeners ou par redéfinition de «callbacks»

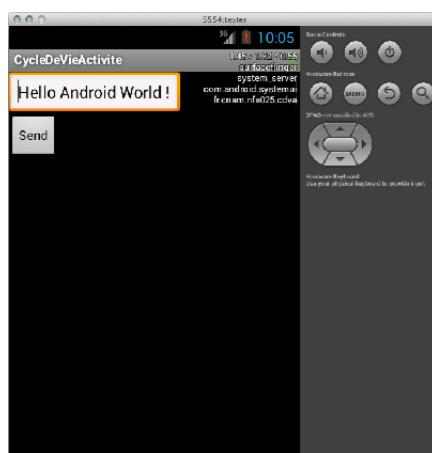


```
1 public void actionOnSendBtn(View view){  
2     Log.d(APP_TAG, "CycleDeVieActivite actionSendBtn!", null);  
3     EditText msg_text_widget = (EditText)this.findViewById(R.id.main_msg_te);  
4     Toast.makeText(this, msg_text_widget.getText(), 2000).show();  
5 }
```

```
1 <?xml version="1.0" encoding="utf-8"?>  
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
3     android:orientation="vertical"  
4     android:layout_width="fill_parent"  
5     android:layout_height="fill_parent"  
6     >  
7     <EditText  
8         android:layout_width="wrap_content"  
9         android:layout_height="wrap_content"  
10        android:id="@+id/main_msg_te"  
11        android:text="@string/launchedMessage"  
12    >/<Button  
13        android:layout_width="wrap_content"  
14        android:layout_height="wrap_content"  
15        android:id="@+id/sendBtn"  
16        android:text="SendBtn"  
17    >  
18    </EditText>  
19 </LinearLayout>
```

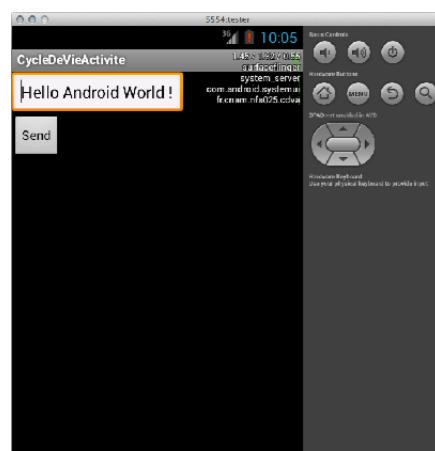
Interface Graphique

- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches
- Interaction à travers des EventListeners ou par redéfinition de «callbacks»



Interface Graphique

- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches
- Interaction à travers des EventListeners ou par redéfinition de «callbacks»



```
1  EditText msg_text_widget;
2  @Override
3  public void onCreate(Bundle savedInstanceState){
4      ...
5      msg_text_widget = new EditText(this);
6      msg_text_widget.setText("Hello Android World !");
7      Button bt = new Button(this);
8      bt.setText("Send");
9      bt.setOnClickListener(new View.OnClickListener() {
10          public void onClick(View arg0) {
11              actionSendBtn(arg0);
12          }
13      });
14      LinearLayout lt = new LinearLayout(this);
15      ...
16  }
17  public void actionSendBtn(View view){
18      Log.d(APP_TAG, "CycleDeVieActivite actionSendBtn!", null);
19      Toast.makeText(this, msg_text_widget.getText(), 2000).show();
20  }
21 }
```

Interface Graphique

7

- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches
- Interaction à travers des EventListeners ou par redéfinition de «callbacks»

Interface Graphique

7

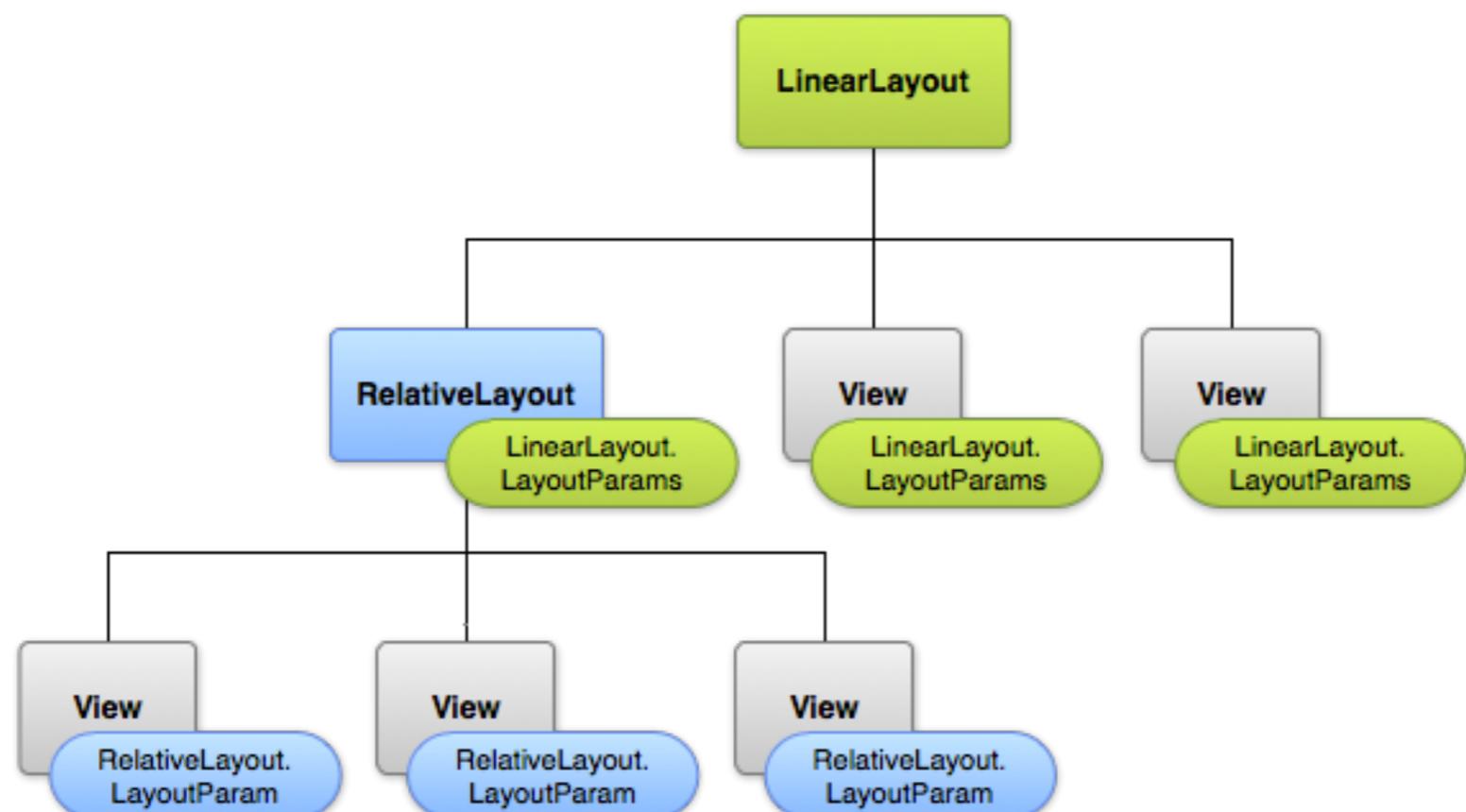
- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches
- Interaction à travers des EventListeners ou par redéfinition de «callbacks»
- La classe R de l'environnement Android est mise à jour automatiquement par le processus de compilation d'un projet Android (pas toujours synchronisée avec l'IDE)

Les vues Android

8

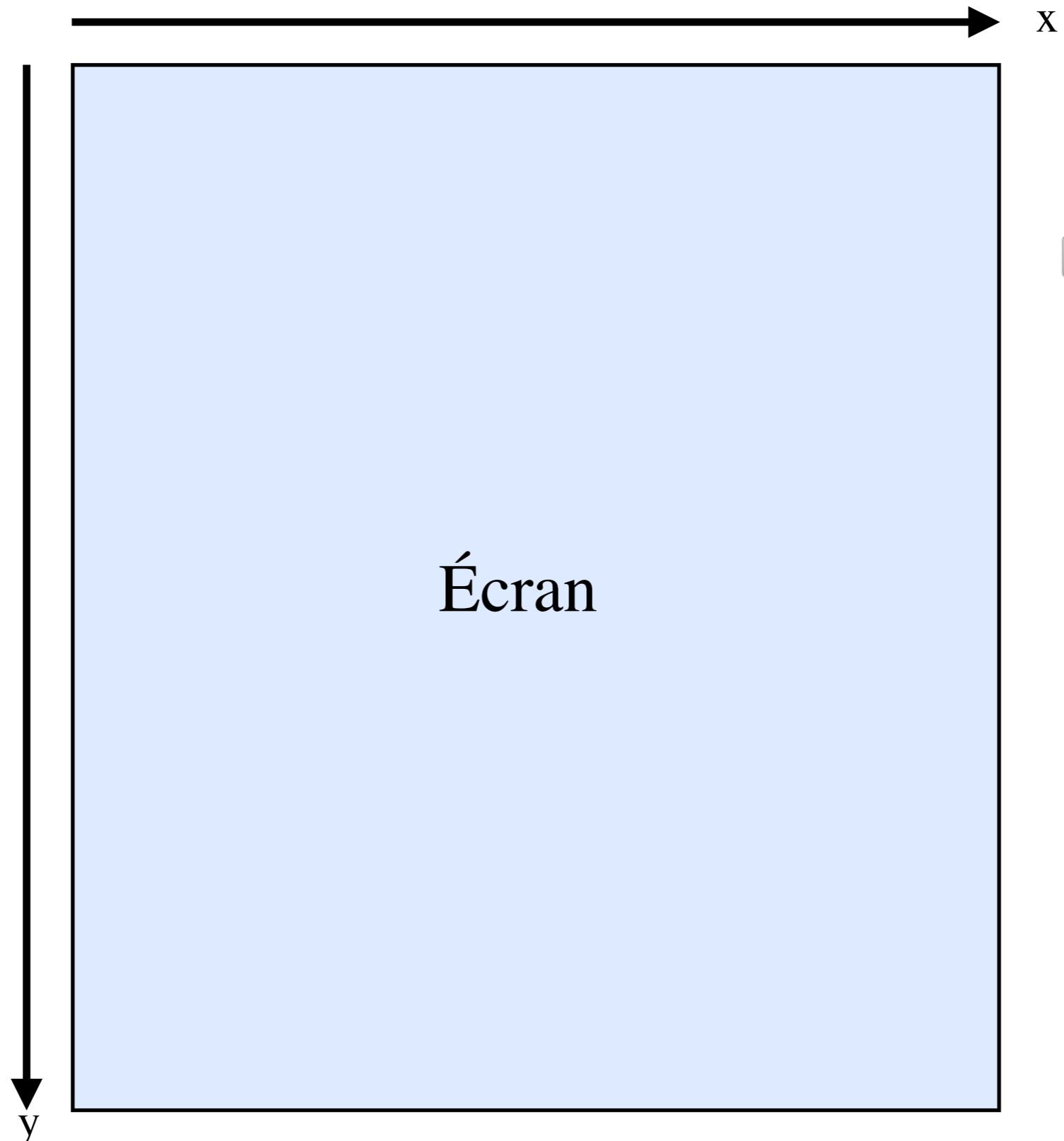
- Chaque objet graphique (appelé vue) possède une balise spécifique dans la description *XML* utilisée par Android et un ensemble d'attributs correspondant à des propriétés de la vue ou à son intégration dans un **ViewGroup**.
- Mécanisme de gestion de la disposition des *widgets* à l'écran : la notion de *Layout* qui permet de manière calculatoire de placer les *widgets* (et de s'adapter automatiquement aux capacités d'affichage du matériel). Ce système traite 2 problèmes :
 - la résolution
 - le format du plan d'affichage (aspect ratio)

Mise en page



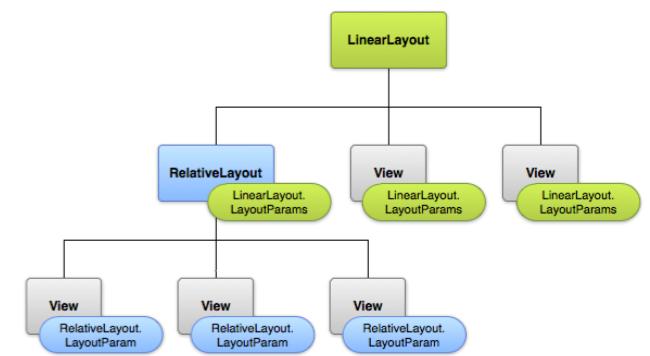
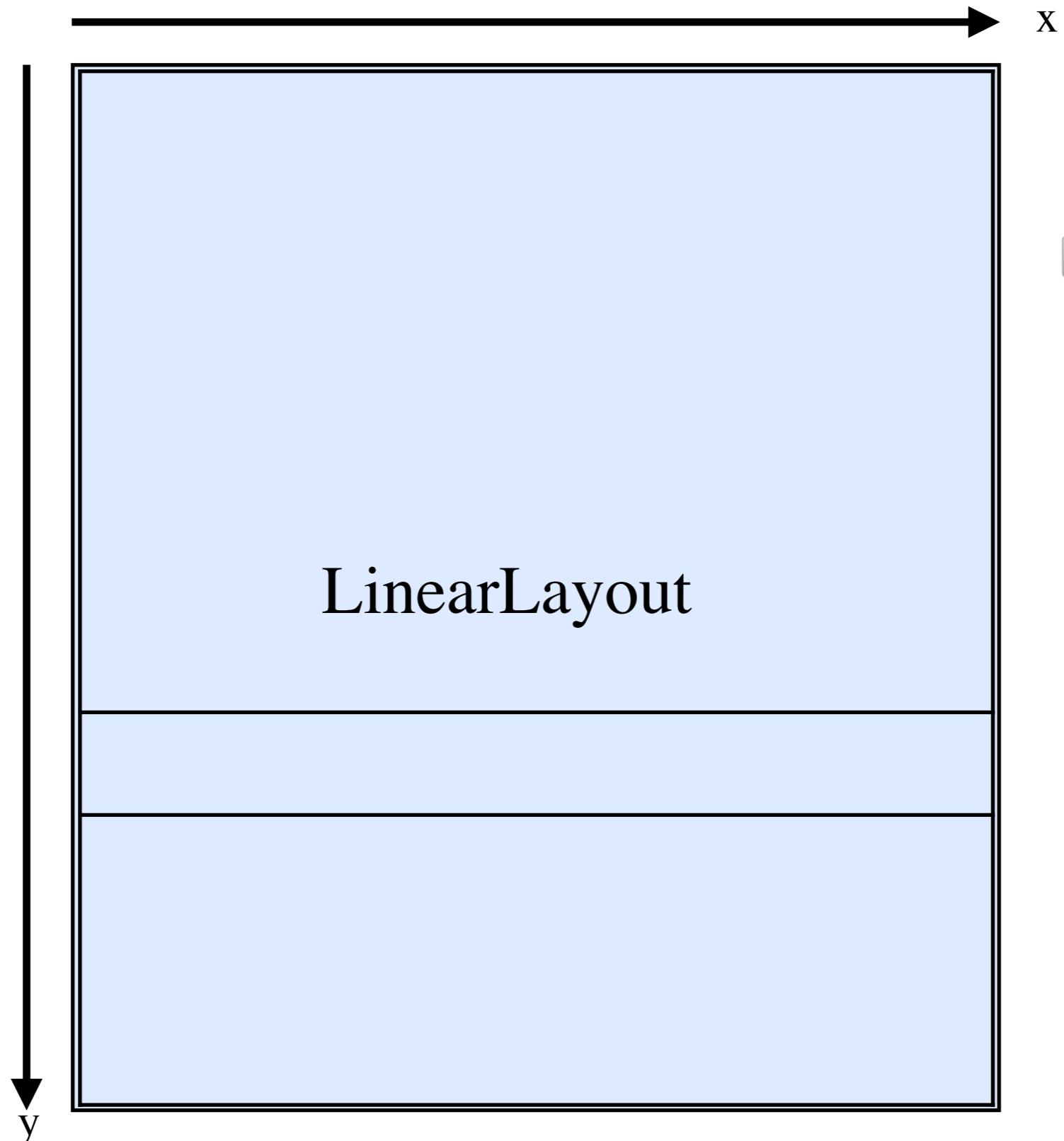
Mise en page

9

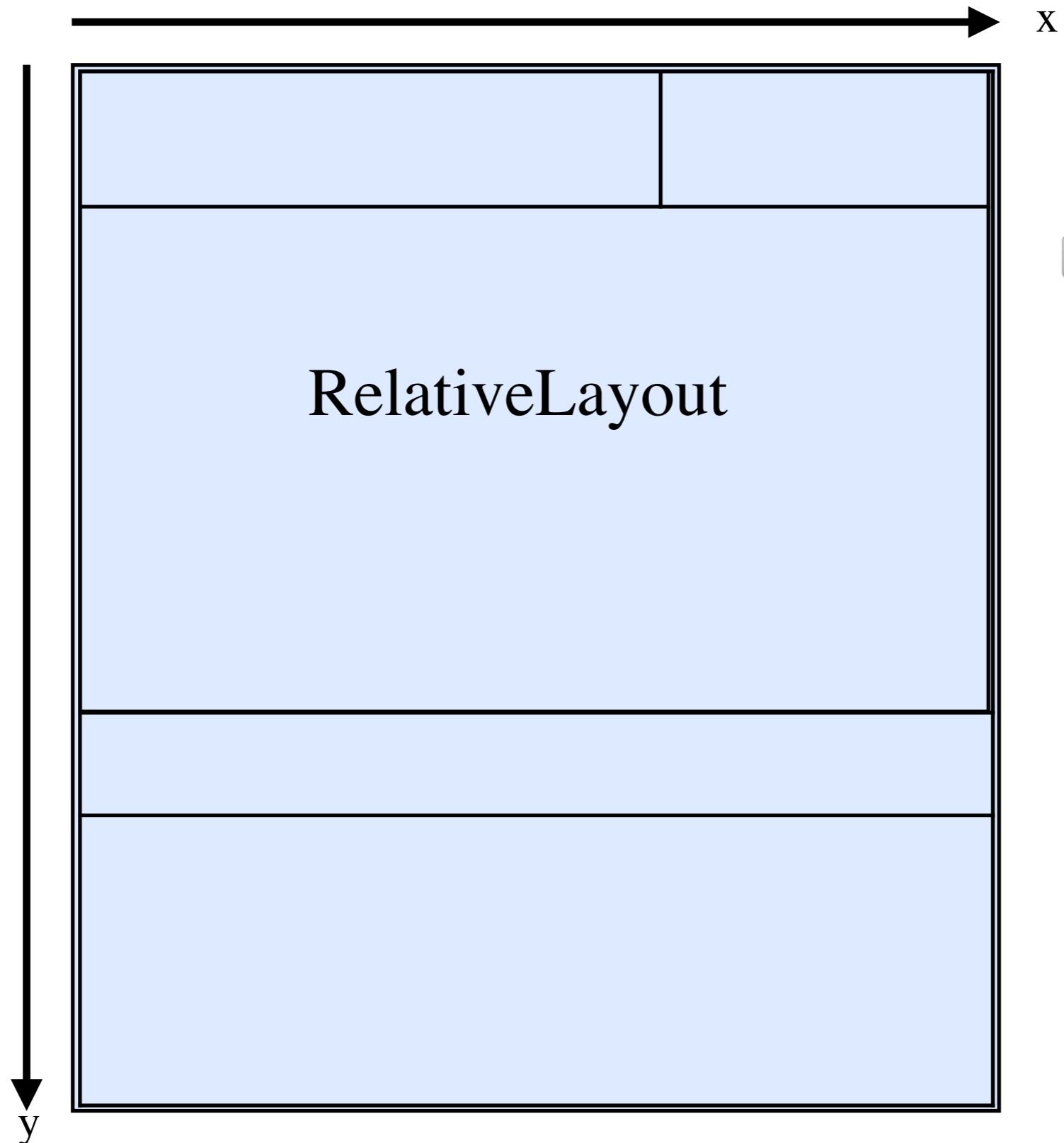


Mise en page

9



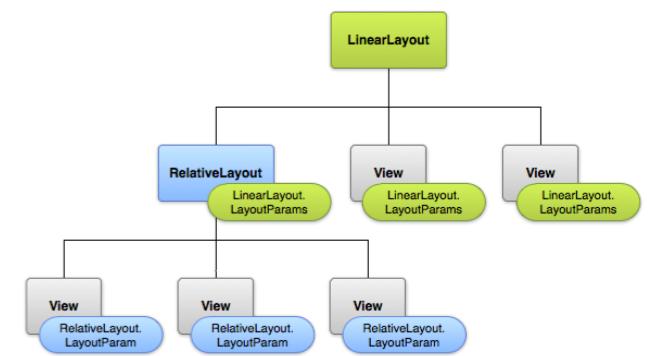
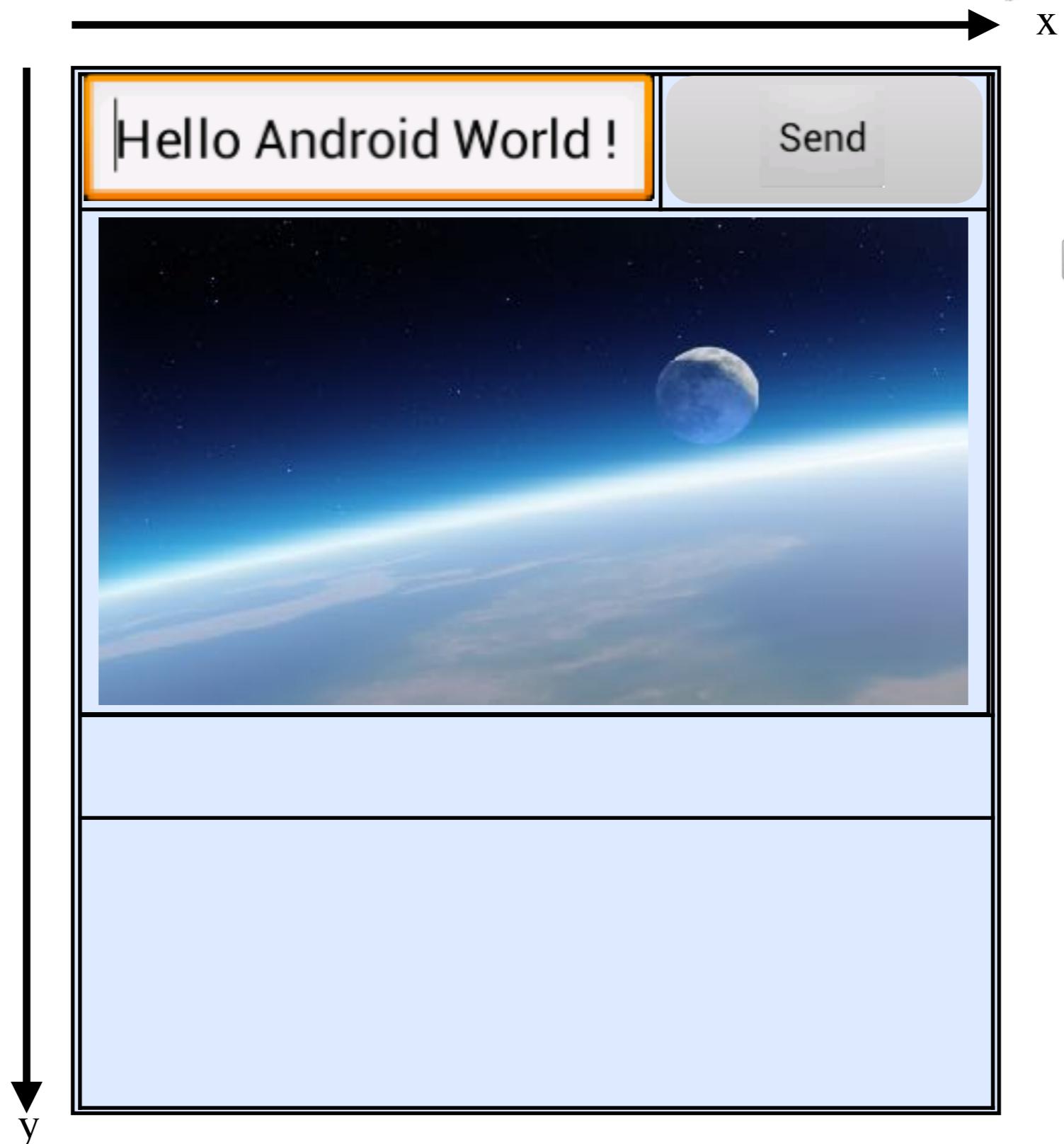
Mise en page



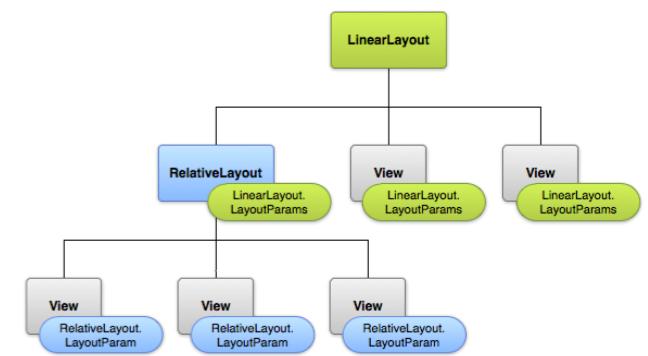
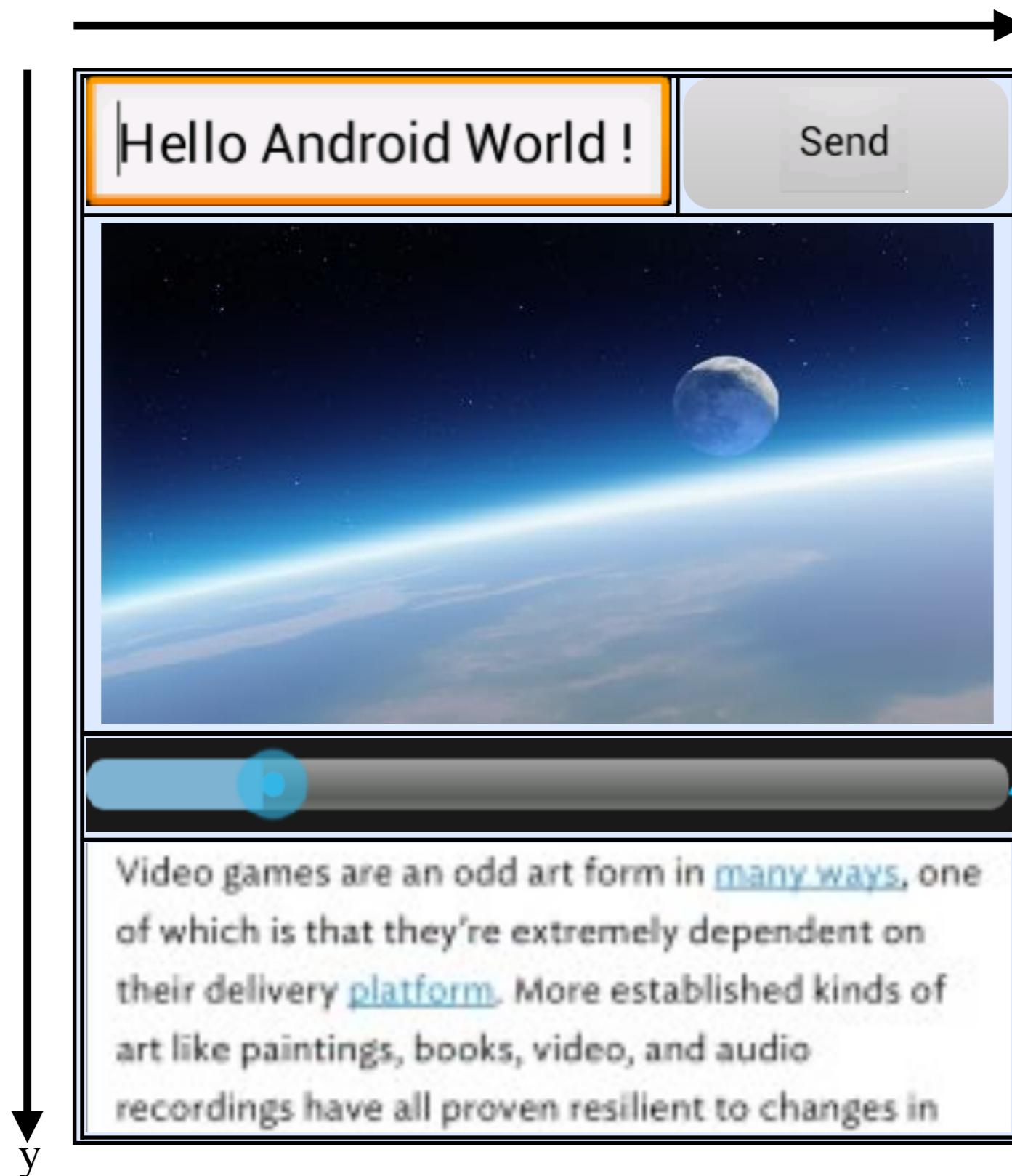
RelativeLayout

Mise en page

9



Mise en page



Les Fragments

10

- Introduits avec la version 3 d’Android, il s’agit de composants d’interface graphique, gérant des portions d’interface réutilisables et gérés par transactions.
- Ils gèrent le cycle de vie d’une petite hiérarchie de vues constituant un tout cohérent et réutilisable.
 - un *fragment* est toujours utilisé dans le contexte d’une activité (seule une activité peut gérer l’*IHM*)
 - un *fragment* peut être ajouté, modifié, remplacé en cours d’exécution de l’activité et une pile de navigation entre *fragments* peut-être maintenue par le développeur.

Les Fragments

11

- Pour créer un nouveau Fragment il faut construire une sous classe de la classe `android.app.Fragment`.
- Il convient ensuite d'implanter les méthodes de son cycle de vie :
 - `onAttach()`
 - `onCreate()`
 - `onCreateView()`
 - `onStart()`
 - `onResume()`
 - `onPause()`
 - `onStop()`
 - `onDestroyView()`
 - `onDestroy()`
 - `onDetach()`

Les Fragments

12

- La gestion des *fragments* est réalisée par le système (**FragmentManager**), mais le contrôle du développeur est plus important que pour un composant applicatif comme une activité.
- Les interactions avec le gestionnaire de *Fragments* passent par la notion de transaction (**FragmentTransaction**).
 - On démarre la transaction : **beginTransaction()**
 - On effectue des transformations : ajout (**add**), suppression (**remove**), remplacement (**replace**)...
 - On termine la transaction : **commit()**.