

INFO0406 - Programmation sur Smartphone





Sommaire

- Interface d'une applications Android
- Layout d'applications
- Composants graphiques d'une application



Rappels

- Composants d'une application :
 - Activity
 - Service
 - BroadcastReceiver
 - ContentProvider
 - Intent
 - Widgets
- Fichier Manifest : AndroidManifest.xml
- Cycle de vie d'une Activity : Running, Paused & Stopped
- La Back Stack



Interface d'une Applications Android



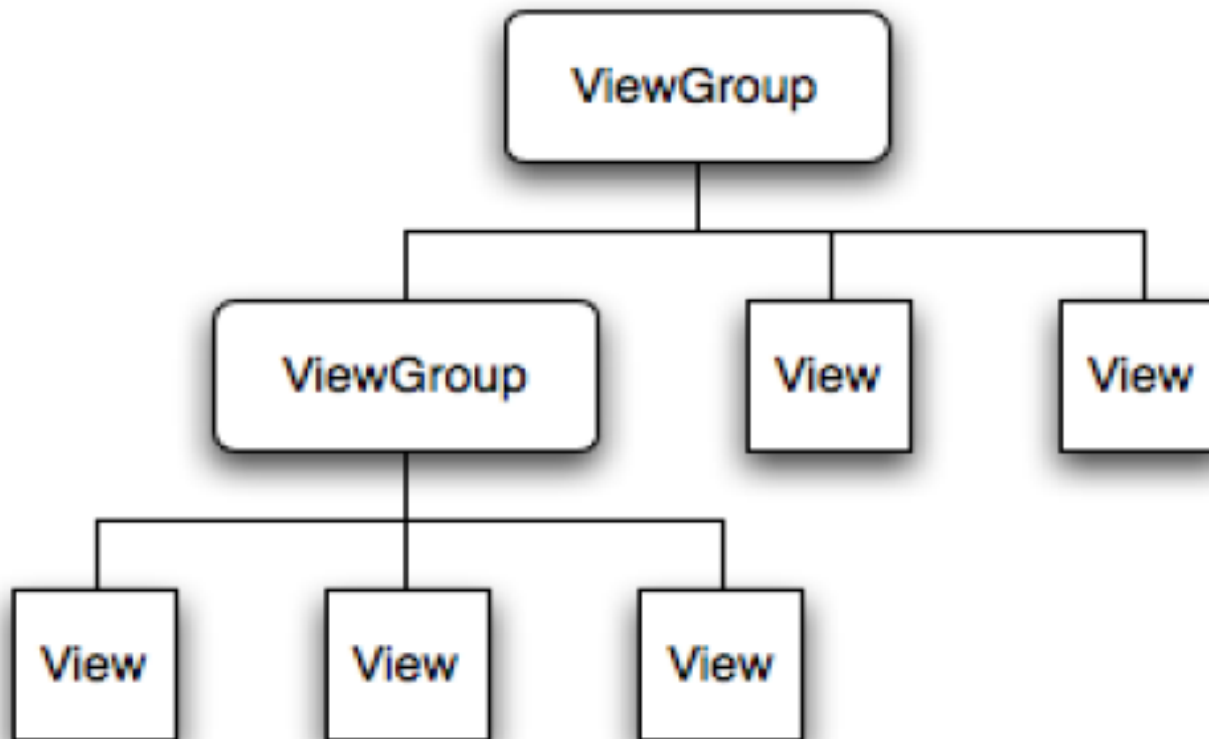


La Classe VIEW

- La classe View est la classe de base de l'interface graphique.
- Une View occupe une zone rectangulaire sur l'écran et est responsable du dessin et de la gestion des événements d'interaction.
- View est la classe de base des widgets qui sont utilisés pour créer des interfaces graphiques (boutons, champ texte, liste, image, etc.).
- La classe ViewGroup est la classe de base pour les Layouts qui sont les conteneurs qui contiennent les autres Views ou autres ViewGroups et définissent les propriétés de mise en forme.



La Classe VIEW





La Classe VIEW

- Les View d'une fenêtre sont arrangées sous la forme d'un arbre.
- Vous pouvez ajouter des Views depuis le code ou en spécifiant un arbre de Views depuis un ou plusieurs fichiers XML Layout.
- Quelques opérations pouvant être effectuées une fois les Views créées:
 - **Set properties** : par exemple le texte d'une TextView. Les propriétés connues à la compilation peuvent être définies dans le fichier XML layout.
 - **Set focus** : Pour forcer le focus sur une View spécifique appelez `requestFocus()`.
 - **Set up listeners** : Views permettent aux clients d'affecter des listeners qui seront notifiés quand un élément intéressant pour la View sera émis. (ex: un bouton expose un listener pour être notifié des clics).
 - **Set visibility** : Vous pouvez masquer ou rendre visible des Views en utilisant `setVisibility(int)`.



La Classe VIEW

- Pour implémenter une View customisée, vous allez surcharger des méthodes comme *onDraw(android.graphics.Canvas)* ou le constructeur
- Évènements customisables :
 - CREATION
 - *onFinishInflate()* : Appelée après qu'une View et ses sous classes soient créées d'un XML.
 - LAYOUT
 - *onMeasure()* : Appelée pour déterminer la taille d'une View et de ses enfants.
 - *onLayout()* : Appelée quand une vue repositionne l'ensemble de ses fils.
 - *onSizeChanged()* : Appelée quand la taille de la vue change.
 - EVENT PROCESSING
 - *onKeyDown()* : Appelée quand une touche est pressée.
 - *onKeyUp()* : Appelée quand une touche est relâchée.
 - *onTrackballEvent()* : Appelée quand une action est réalisée sur la trackball.
 - *onTouchEvent()* : Appelée quand une action est réalisée sur l'écran tactile.



La Classe VIEW

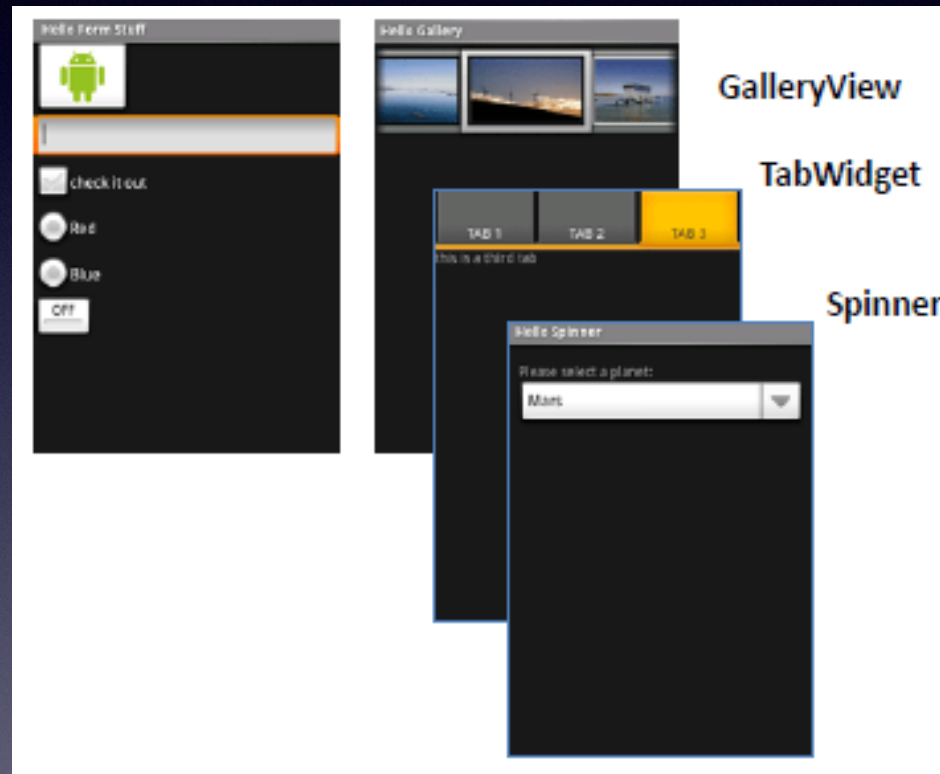
- Évènements customisables (suite) :
 - FOCUS
 - *onFocusChanged()* : Appelée quand la View perd le focus.
 - *onWindowFocusChanged()* : Appelée quand la fenêtre contenant la View perd ou gagne le focus.
 - ATTACHING
 - *onAttachedToWindow()* : Appelée quand la View est attachée à la fenêtre.
 - *onDetachedFromWindow()* : Appelée quand la View est détachée à la fenêtre.
 - *onWindowVisibilityChanged()* : Appelée quand la visibilité de la fenêtre contenant la View a changé.
 - DRAWING
 - *onDraw()* : View doit dessiner son contenu.



Quelques exemples de Views



DatePicker



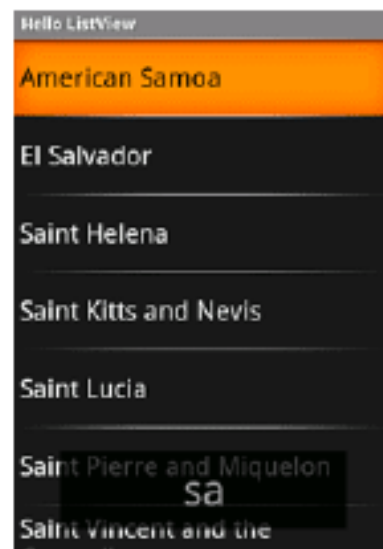
Widgets permettant la réalisation de formulaires ou d'interfaces graphiques.



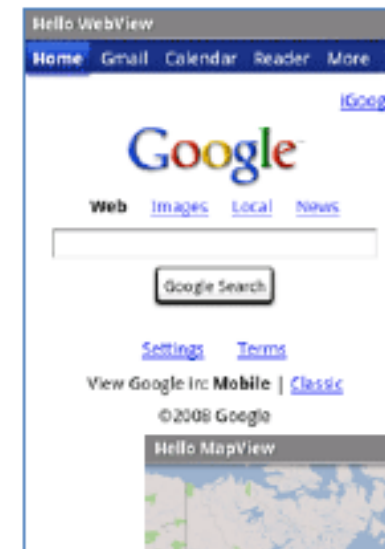
Quelques exemples de Views



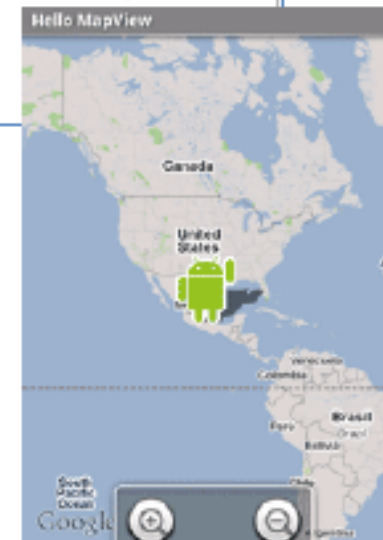
AutoCompleteTextView



ListView



WebView



MapView



Les classes conteneurs : Layout

- Déclaration d'un *Layout*
 - Un *Layout* est l'architecture d'une interface utilisateur d'une *Activity*.
 - Le *Layout* définit la structure de tous les éléments contenus qui apparaissent à l'utilisateur.
 - Les *Layouts* peuvent être déclarés :
 - Sous forme XML:
 - Android fournit un vocabulaire XML simple qui correspond aux classes et sous-classes *Views* comme les *widgets* et les *Layouts*.
 - A l'exécution:
 - Votre application peut créer des *Views* et *ViewGroup* (*Layouts*) et manipuler leurs propriétés à l'exécution.



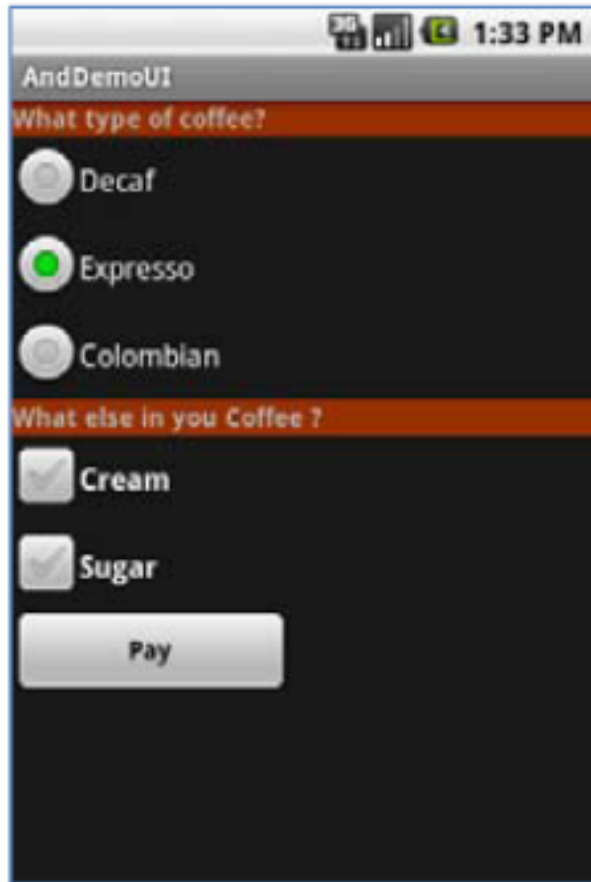
Les classes conteneurs : Layout

- Android propose un *LinearLayout* par défaut et d'autres *Layouts*.
- Le *LinearLayout* offre un modèle similaire à Java–Swing *Box–Layout*.
- La stratégie pour avoir l'interface graphique désirée est de composer avec les *Layouts* nécessaires.
- Android offre d'autres organisations:
 - *LinearLayout* le modèle de « boîtes »
 - *RelativeLayout* un modèle basé sur les règles
 - *TableLayout* le modèle basé sur une grille
 - *ScrollView*, un conteneur permettant d'assister les conteneurs avec une fonctionnalité de scroll.



Les classes conteneurs : `FrameLayout`

- *FrameLayout* est le *Layout* le plus simple d'Android qui attache chaque fils au coin haut gauche en empilant les Views l'une sur l'autre.





Les classes Layout : LinearLayout

- *LinearLayout* est un modèle de type box où les widgets et les conteneurs sont alignés en colonne ou en ligne, les uns après les autres
- Pour paramétrer un *LinearLayout*, vous avez 5 propriétés:
 - Orientation
 - Fill model
 - Weight
 - Gravity
 - Padding




LinearLayout : Orientation

- Indique si le *LinearLayout* représente une ligne ou une colonne.
- Ajouter ***android:orientation*** à votre XML décrivant le *LinearLayout* avec ***horizontal*** pour une ligne et ***vertical*** pour une colonne.
- L'orientation peut être modifiée à l'exécution par *setOrientation()*




LinearLayout : Orientation

vertical



→



```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:id="@+id/myLinearLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ff0033cc"
    android:padding="4px"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" >
    <TextView
        android:id="@+id/labelUserName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#ffff0066"
        android:text="User Name"
        android:textSize="16sp"
        android:textStyle="bold"
        android:textColor="#ff000000"
    >
    </TextView>
    <EditText
        android:id="@+id/ediName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"
    >
    </EditText>
    <Button
        android:id="@+id/btnGo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Go"
        android:textStyle="bold"
    >
    </Button>
</LinearLayout>

```




LinearLayout : Fill Model

- Les Widgets ont une taille naturelle basée sur leur propre contenu.
- Lorsque leur taille ne correspond pas exactement à la largeur de l'écran, nous pouvons avoir à traiter l'espace restant.
- Tous les widgets d'un *LinearLayout* doivent fournir des attributs dimensionnels ***android:layout_width*** et ***android:layout_height*** pour faciliter la gestion de l'espace vide.
- Les valeurs utilisées pour définir la hauteur et la largeur sont:
 - Une valeur précise *100px* pour indiquer que le widget prendra *100px*.
 - **wrap_content** : le widget doit remplir l'espace, à moins que ce soit trop grand, dans ce cas Android peut utiliser *word-wrap* pour le faire rentrer.
 - **fill_parent** : le widget doit remplir tout l'espace disponible dans son conteneur, après tous les autres widgets sont pris en charge.



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:id="@+id/myLinearLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ff0033cc"
    android:padding="4px"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android"
>
    <TextView
        android:id="@+id/labelUserName"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#ffff0066"
        android:text="User Name"
        android:textSize="16sp"
        android:textStyle="bold"
        android:textColor="#ff000000"
    >
    </TextView>
    <EditText
        android:id="@+id/ediName"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="18sp"
    >
    </EditText>
    <Button
        android:id="@+id/btnGo"
        android:layout_width="125px"
        android:layout_height="wrap_content"
        android:text="Go"
        android:textStyle="bold"
    >
    </Button>
</LinearLayout>
```

Row-wise

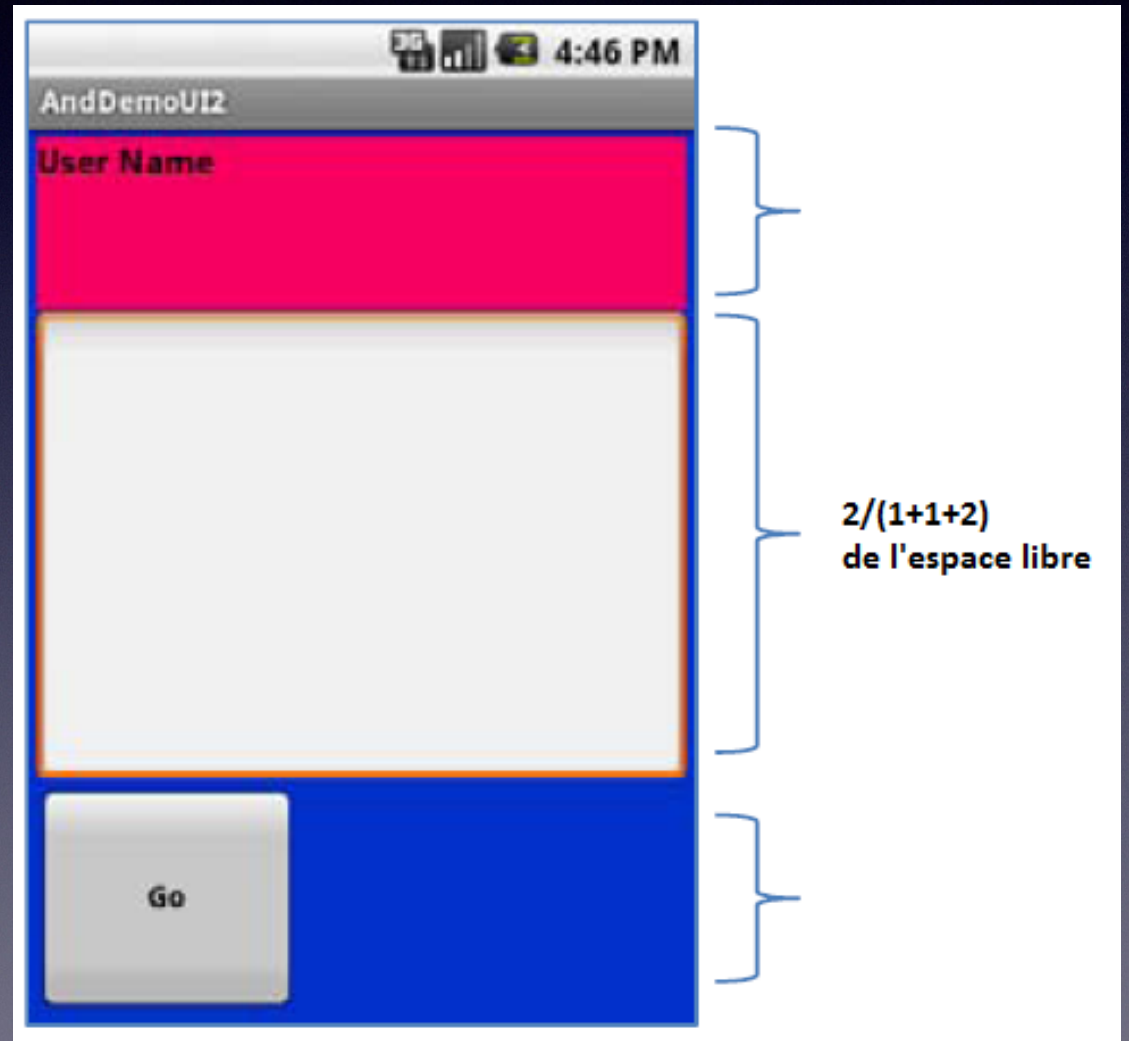
Use all the row

Specific size: 125px



LinearLayout : Weight

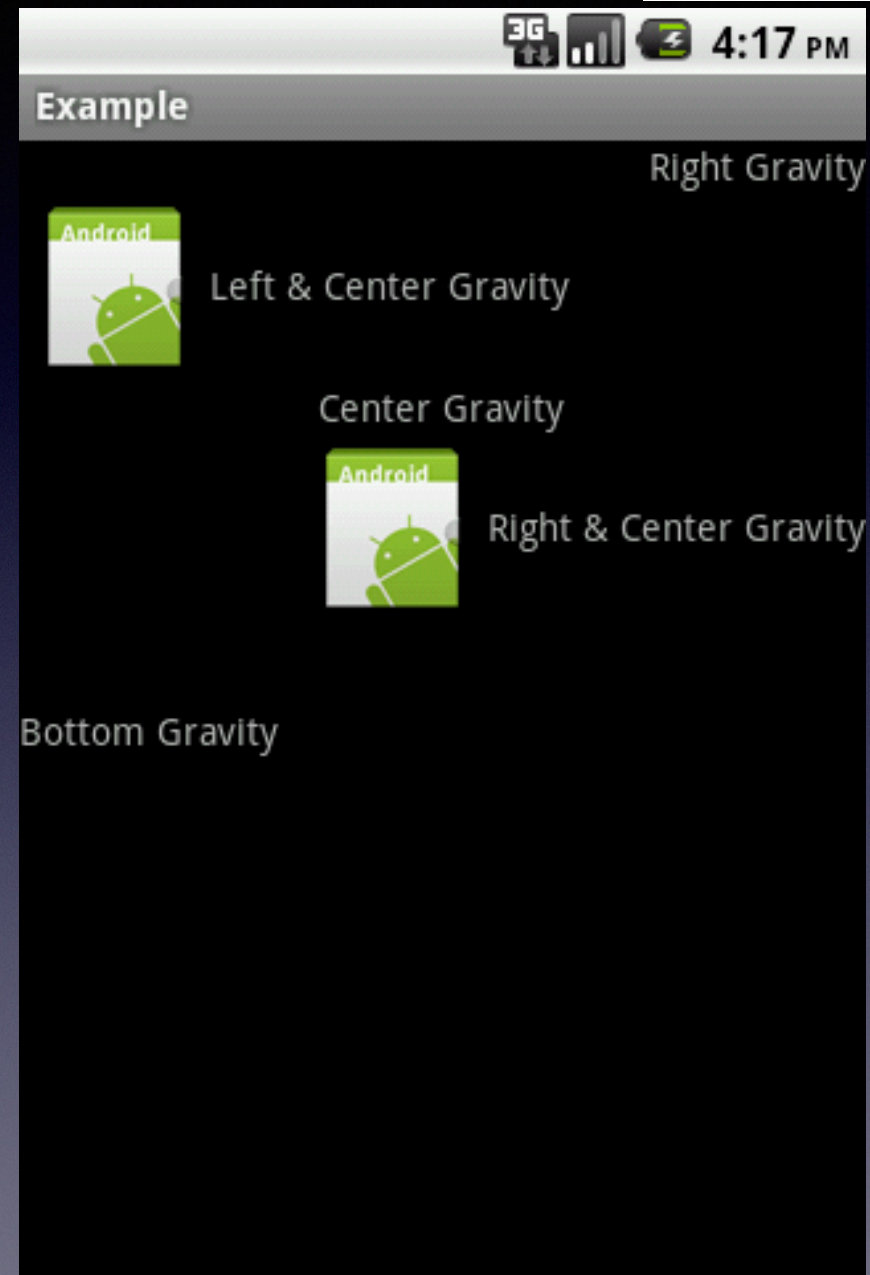
- *Weight* est utilisé pour assigner de l'espace proportionnellement au poids définis entre les widgets dans le conteneur.
- Affectez à `android:layout_weight` une valeur (1, 2, 3, 4 ...) pour indiquer quelle proportion de l'espace libre doit être dédiée à ce widget.
- Pour l'exemple précédent :
 - TextView et Boutons : **`android:layout_weight = "1"`**
 - EditionTexte : **`android:layout_weight = "2"`**





LinearLayout : Gravity

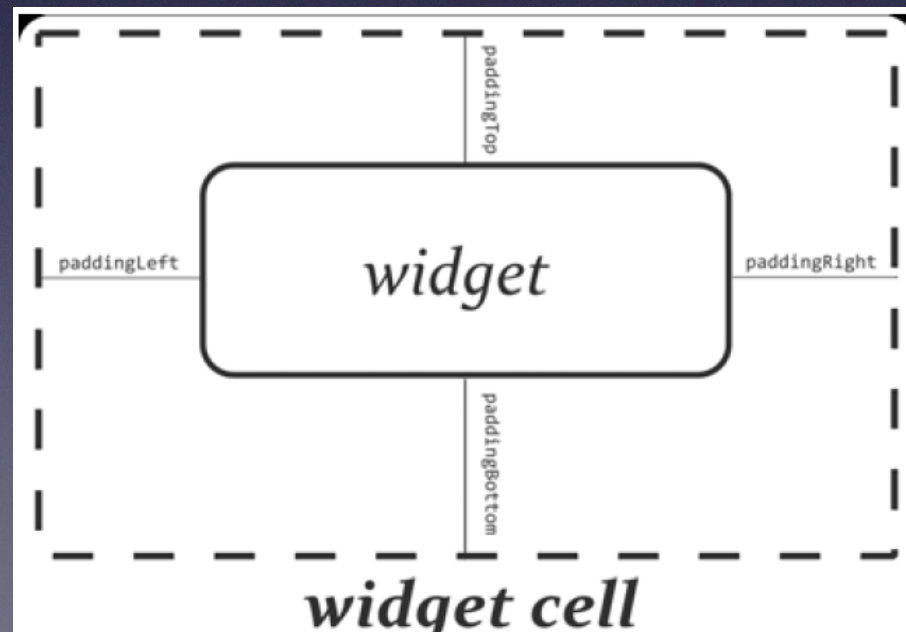
- **Gravity** indique comment un widget sera aligné sur l'écran.
- Par défaut, les widgets sont alignés en haut à gauche
- Dans le XML utilisez **`android:layout_gravity="..."`** avec pour valeurs:
 - top / bottom
 - left / right
 - center
 - fill
 - center_vertical / center_horizontal
 - fill_vertical / fill_horizontal
 - clip-vertical / clip_horizontal





LinearLayout : Padding

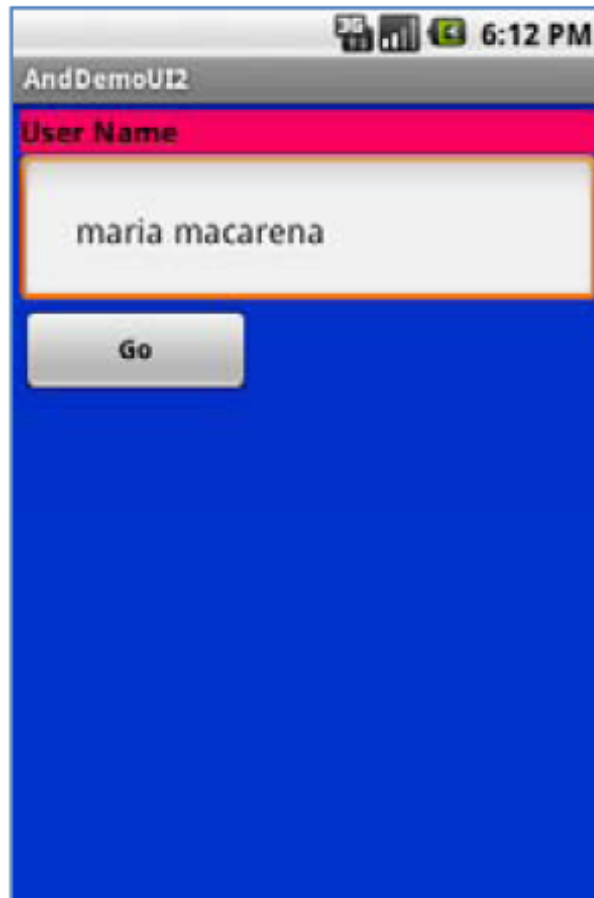
- Par défaut, les widgets sont serrés les uns à côté des autres.
- Pour augmenter l'espace entre les widgets, on utilise la propriété **padding** du XML ou en appelant `setPadding()` lors de l'exécution.
- Le **padding** spécifie combien d'espace il y a entre les limites de "La cellule" du widget et le contenu réel de widgets.
- Note: **padding** est analogue aux marges d'un document.





LinearLayout : Exemple Padding

- Ajout de 30 pixels de padding tout autour de l'editText:



```
<EditText  
    android:id="@+id/ediName"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:textSize="18sp"  
  
    android:padding="30px"  
  
>  
</EditText>  
...
```




Les classes Layout : RelativeLayout

- **RelativeLayout** place les widgets les uns par rapport aux autres.
- Paramètres de positionnement relatif au conteneur parent :
 - **android:layout_alignParentTop** : Cette option permet de préciser si le haut de l'élément doit être aligné avec celui de son conteneur.
 - **android:layout_alignParentBottom** : Cette option permet de préciser si le bas de l'élément doit être aligné avec celui de son conteneur.
 - **android:layout_alignParentLeft** : Cette option permet de préciser si le côté gauche de l'élément doit être aligné avec celui de son conteneur.
 - **android:layout_alignParentRight** : Cette option permet de préciser si le côté droit de l'élément doit être aligné avec celui de son conteneur.
 - **android:layout_centerHorizontal** : Indique si l'élément doit être centré horizontalement dans son conteneur.
 - **android:layout_centerVertical** : Indique si l'élément doit être centré verticalement dans son conteneur.
 - **android:layout_centerInParent** : Vous permet d'indiquer que l'élément doit être centré horizontalement et verticalement dans le conteneur.



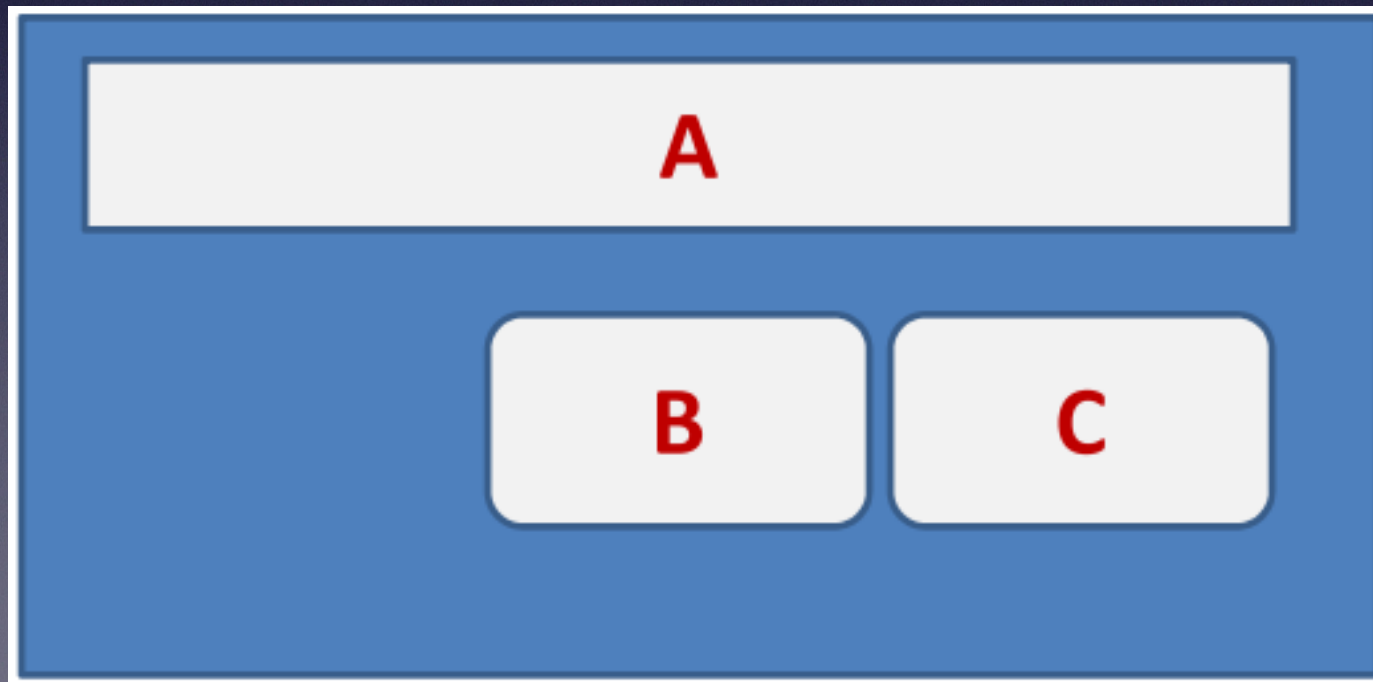
Les classes Layout : RelativeLayout

- Position relative aux autres éléments :
 - **android:layout_above** : Indique que l'élément sera placé au-dessus de celui indiqué par son id.
 - **android:layout_below** : Indique que l'élément sera placé en dessous de celui indiqué par son id.
 - **android:layout_toLeftOf** : Indique que l'élément sera placé à gauche de celui indiqué par son id.
 - **android:layout_toRightOf** : Indique que l'élément sera placé à droite de celui indiqué par son id.
 - **android:layout_alignTop** : Indique que le haut de notre élément est aligné avec le haut de l'élément indiqué.
 - **android:layout_alignBottom** : Indique que le bas de notre élément est aligné avec le bas de l'élément indiqué.
 - **android:layout_alignLeft** : Indique que le côté gauche de notre élément est aligné avec le côté gauche de l'élément indiqué.
 - **android:layout_alignRight** : Indique que le côté droit de notre élément est aligné avec le côté droit de l'élément indiqué.
 - **android:layout_alignBaseline** : Indique que les lignes de base des 2 éléments sont alignées.



Les classes Layout : RelativeLayout

- Exemple :
 - A est positionné en haut du conteneur
 - C est sous A à sa droite
 - B est sous A à gauche de C





Les classes Layout : RelativeLayout

- Code de l'Exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:id="@+id/myRelativeLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ff000099"
    xmlns:android="http://schemas.android.com/apk/res/android"
    >

    <TextView
        android:id="@+id/lblUserName"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#ffff0066"
        android:text="User Name"
        android:textStyle="bold"
        android:textColor="#ff000000"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
    >

        <EditText
            android:id="@+id/ediUserName"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_below="@+id/lblUserName"
            android:layout_alignParentLeft="true"
            android:layout_alignLeft="@+id/myRelativeLayout"
            android:padding="20px"
        >

        <Button
            android:id="@+id/btnGo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@+id/ediUserName"
            android:layout_alignRight="@+id/ediUserName"
            android:text="Go"
            android:textStyle="bold"
        >

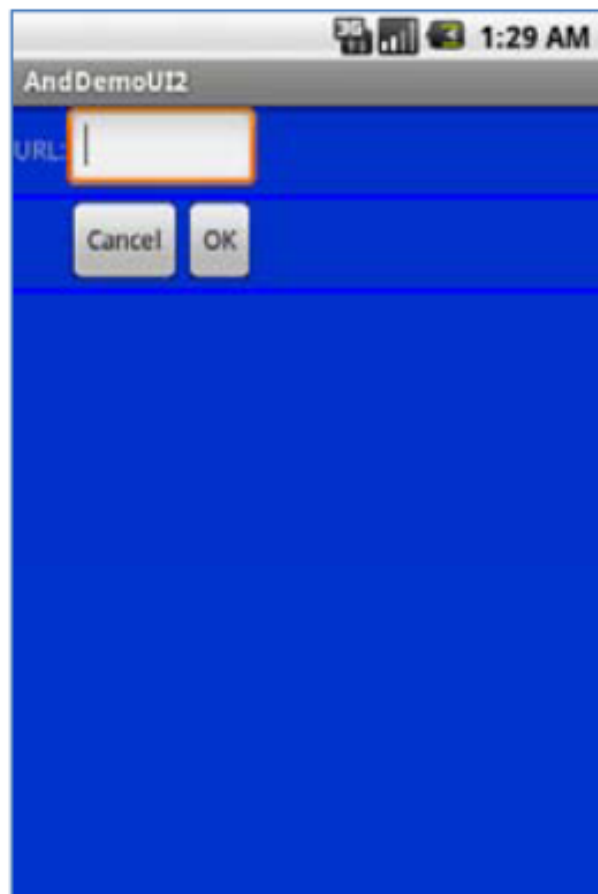
        <Button
            android:id="@+id/btnCancel"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_toLeftOf="@+id/btnGo"
            android:layout_below="@+id/ediUserName"
            android:text="Cancel"
            android:textStyle="bold"
        >
    </RelativeLayout>
```





Les classes Layout : **TableLayout**

- **TableLayout** permet de positionner les widgets dans une grille.
- Les colonnes peuvent s'étirer ou se rétrécir en fonction du widget placé dans la cellule.
- **TableLayout** fonctionne avec des **TableRows**
- **TableLayout** contrôle le fonctionnement global du conteneur et des widgets positionnés dans une ou plusieurs **TableRows**
- Les lignes sont déclarées en plaçant des widgets comme enfant de **TableRow** dans le **TableLayout**.
- Le nombre de colonnes est déterminé par Android, on utilise la propriété **android:layout_span** pour étaler un widget sur plusieurs colonnes.



```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    android:id="@+id/myTableLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ff0033cc"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android"
>
    <TableRow>
        <TextView
            android:text="URL: " />
        <EditText android:id="@+id/ediUrl"
            android:layout_span="3"/>
        </TableRow>
        <View
            android:layout_height="3px"
            android:background="#0000FF" />
        <TableRow>
            <Button android:id="@+id/cancel"
                android:layout_column="2"
                android:text="Cancel" />
            <Button android:id="@+id/ok"
                android:text="OK" />
        </TableRow>
        <View
            android:layout_height="3px"
            android:background="#0000FF" />
    </TableLayout>
```

Stretch up to column 3

Skip column: 1


```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tableLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
```

```
<!-- 2 columns -->
```

```
<TableRow
    android:id="@+id/tableRow1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="5dip" >
```

```
<TextView
```

```
    android:id="@+id/textView1"
    android:text="Column 1"
    android:textAppearance=
```

```
"?android:attr/textAppearanceLarge" />
```

```
<Button
```

```
    android:id="@+id/button1"
    android:text="Column 2" />
```

```
</TableRow>
```

```
<!-- edittext span 2 column -->
```

```
<TableRow
    android:id="@+id/tableRow2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="5dip" >
```

```
<EditText
```

```
    android:id="@+id/editText1"
    android:layout_span="2"
    android:text="Column 1 & 2" />
```

```
</TableRow>
```

```
<!-- just draw a red line -->
```

```
<View
    android:layout_height="2dip"
    android:background="#FF0000" />
```

```
<Button
```

```
    android:id="@+id/button3"
    android:text="Column 3" />
```

```
</TableRow>
```

```
<!-- display this button in 3rd column via layout_column(zero based) -->
```

```
<TableRow
    android:id="@+id/tableRow4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="5dip" >
```

```
<Button
```

```
    android:id="@+id/button4"
    android:layout_column="2"
    android:text="Column 3" />
```

```
</TableRow>
```

```
<!-- display this button in 2nd column via layout_column(zero based) -->
```

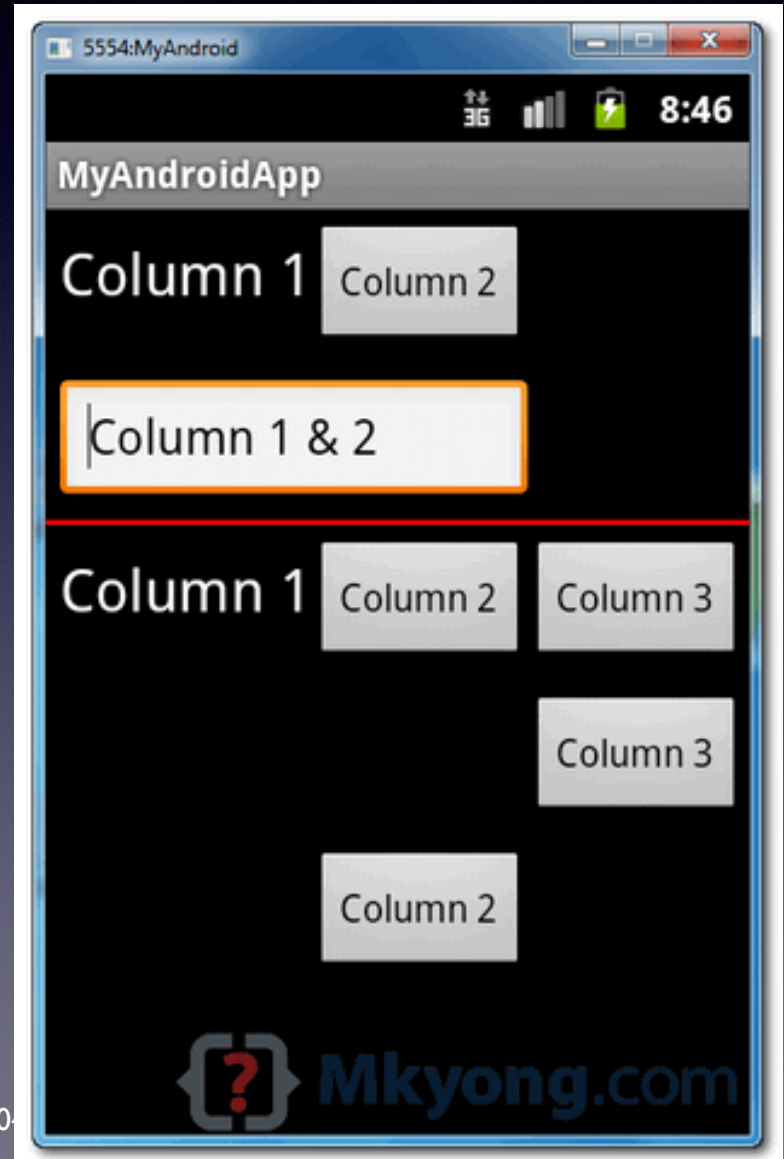
```
<TableRow
    android:id="@+id/tableRow5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="5dip" >
```

```
<Button
```

```
    android:id="@+id/button5"
    android:layout_column="1"
    android:text="Column 2" />
```

```
</TableRow>
```

```
</TableLayout>
```





Les classes Layout : ScrollView

- **ScrollView** vous permet de traiter la cas où vous avez plus de data à afficher que ce qui peut l'être à l'écran.
- Les datas sont alors accessibles par sliding (coulissant) ou scrolling (défilement).



```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
    android:id="@+id/widget28"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ff009999"
    xmlns:android="http://schemas.android.com/apk/res/android"
>
    <LinearLayout
        android:id="@+id/myLinearLayoutVertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical"
    >

        <LinearLayout
            android:id="@+id/myLinearLayoutHorizontal1"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:orientation="horizontal"
        >
            <ImageView
                android:id="@+id/myPicture"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:src="@drawable/icon" />

            <TextView
                android:id="@+id/textView1"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="Line1"
                android:textSize="70px" />
            </LinearLayout>

            <TextView
                android:layout_width="fill_parent"
                android:layout_height="6px"
                android:background="#ffccffcc" />
        </LinearLayout>
    </ScrollView>
```

```
<TextView
    android:id="@+id/textView2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Line2"
    android:textSize="70px" />

<TextView
    android:layout_width="fill_parent"
    android:layout_height="6px"
    android:background="#ffccffcc" />

<TextView
    android:id="@+id/textView3"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Line3"
    android:textSize="70px" />

<TextView
    android:layout_width="fill_parent"
    android:layout_height="6px"
    android:background="#ffccffcc" />

<TextView
    android:id="@+id/textView4"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Line4"
    android:textSize="70px" />

<TextView
    android:layout_width="fill_parent"
    android:layout_height="6px"
    android:background="#ffccffcc" />

<TextView
    android:id="@+id/textView5"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Line5"
    android:textSize="70px" />
</LinearLayout>
</ScrollView>
```



```

<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android=
"http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >

    <LinearLayout

        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >

            <Button
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="Button 1" />

            <Button
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="Button 2" />

            <Button
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="Button 3" />

            <Button
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="Button 4" />

            <Button
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="Button 5" />

            <ProgressBar

                android:id="@+id/progressBar1"
                style="?android:attr/progressBarStyleHorizontal"
                android:layout_width="match_parent"
                android:layout_height="wrap_content" />

            <RadioButton
                android:id="@+id/radioButton2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="RadioButton 1" />

            <RadioButton
                android:id="@+id/radioButton1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="RadioButton 2" />

            <ToggleButton
                android:id="@+id/toggleButton1"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="ToggleButton" />

            <Button
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="Button 6" />

            <SeekBar
                android:id="@+id/seekBar1"
                android:layout_width="match_parent"
                android:layout_height="wrap_content" />

            <Button
                android:id="@+id/button3"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="Button 7" />

            <Button
                android:id="@+id/button4"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="Button 8" />

            <CheckBox
                android:id="@+id/checkbox1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="CheckBox" />

        </LinearLayout>

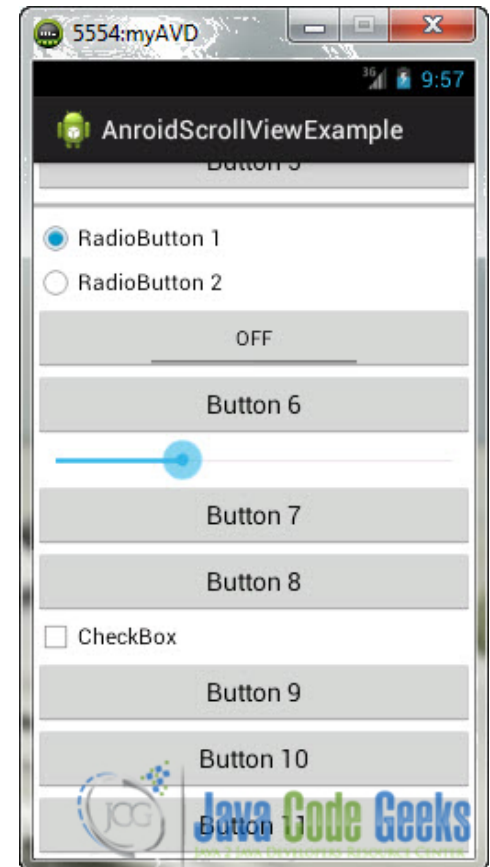
    <Button
        android:id="@+id/button2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 9" />

    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 10" />

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button 11" />

</ScrollView>

```





- Le menu système **Menus systèmes** accessible depuis le bouton home. Votre activity peut définir son propre menu système :

- **onCreateOptionsMenu** : appelée 1 seule fois pour permettre de construire le menu.

- **onPrepareOptionsMenu** : appelée une fois créé avant l'affichage du menu.

- **onOptionsItemSelected** : appelée lorsque l'utilisateur clique sur un item du menu.

```
/* ***** */
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    mMenu = menu;
    return super.onCreateOptionsMenu(menu);
}
/* ***** */

/* ***** */
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    if (mMainView != null) {
        mMainView.RefreshOptionsMenu();
        return true;
    } else return false;
}
/* ***** */

/* ***** */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    if (mMainView != null) {
        mMainView.OptionsItemSelected(item);
        return true;
    } else return false;
}
/* ***** */
```



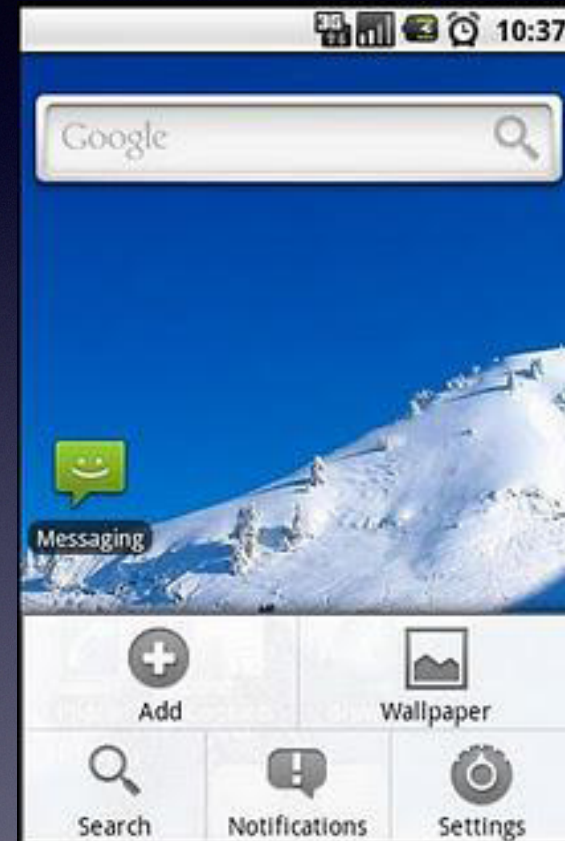

```

/* ***** */
public void RefreshOptionsMenu() {
    if (activityparent.mMenu != null) {
        activityparent.mMenu.clear();
        CreateOptionsMenu(activityparent.mMenu, CurrentStep);
    }
}
/* ***** */

/* ***** */
private boolean CreateOptionsMenu(Menu menu, int p_step) {
    MenuInflater inflater = activityparent.getMenuInflater();
    switch (p_step) {
        case STEP_MENU_MAIN:
            inflater.inflate(R.menu.main_menu, menu);
            return true;
        case STEP_GAME:
            inflater.inflate(R.menu.game_menu, menu);
            activityparent.mMenu.findItem(R.id.undo).setEnabled(undoAvailable);
            return true;
        default:
            return false;
    }
}
/* ***** */

/* ***** */
public boolean OptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.about:
            showAbout();
            return true;
        case R.id.help:
            showHelp();
            return true;
        case R.id.stats:
            showStats();
            return true;
        case R.id.options:
            showOptionsPack();
            return true;
        case R.id.undo:
            undo();
            return true;
        default:
            return false;
    }
}
/* ***** */

```





L'interaction clavier

- Récupérer les actions utilisateurs sur le clavier ou les boutons
- Une *View* vous permet d'être notifié des actions clavier quand une touche est pressée (***onKeyDown***) ou relâchée (***onKeyUp***)
- Le ***Callback*** indique le *keycode* de la touche concernée
- Il indique également un (event de type ***KeyEvent***) contenant le contexte du bouton (pression multiple, ...).
- La classe ***KeyEvent*** contient les constantes de référence (numéro, back, ...)
- Référence Android Developer: <http://developer.android.com/reference/android/view/KeyEvent.html>



L'interaction clavier

```

/* ***** */
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_BACK) {
        switch (CurrentStep) {
            case STEP_CONTACTSMS:
            case STEP_STARTGAMEANIM:
            case STEP_MOREGAMES:
            case STEP_GAME:
                ChangeStep(STEP_MENU_MAIN);
                break;
            case STEP_MENU_MAIN:
            case STEP_LOADING:
                in = false;
                stopSound();
                activityparent.finish();
                break;
        }
        return true;
    }
    return super.onKeyDown(keyCode, event);
}
/* ***** */

```




L'interaction tactile

- Récupérer les actions utilisateurs sur l'écran
- Une View vous permet d'être notifié des actions effectuées sur l'écran tactile (***onTouchEvent(MotionEvent event)***).
- Le MotionEvent transmis permet d'identifier le type d'action et les paramètres associés:
 - DOWN : je pose le doigt
 - UP : je relève le doigt
 - MOVE : je déplace le doigt
- Référence Android Developer: <http://developer.android.com/reference/android/view/MotionEvent.html>



```
/* ***** */  
public boolean onTouchEvent (MotionEvent event) {  
    if (!inPack) {  
        final int action = event.getAction();  
        myx = 1*event.getX();  
        myy = 1*event.getY();  
  
        myx = myx / ((float) getWidth()/((float) myWidth()));  
        myy = myy / ((float) getHeight()/((float) myHeight()));  
  
        switch (action) {  
            case MotionEvent.ACTION_MOVE:  
                onTouchEventMove(event);  
                return true;  
            case MotionEvent.ACTION_UP:  
                onTouchEventUp(event);  
                return true;  
            case MotionEvent.ACTION_DOWN:  
                onTouchEventDown(event);  
                return true;  
        }  
    }  
    return super.onTouchEvent(event);  
}  
/* ***** */
```



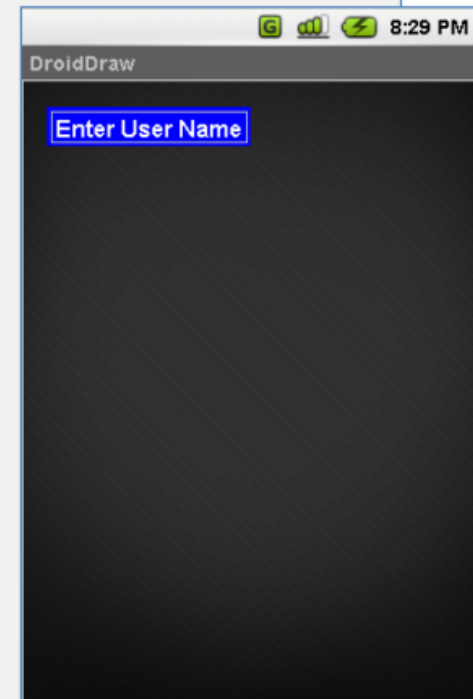

Les classes View :TextView

- **TextView** est utilisé pour afficher un texte non éditable
- Référence Android Developer : <http://developer.android.com/reference/android/widget/TextView.html>



Les classes View · TextView

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
    android:id="@+id/absLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
>
<TextView
    android:id="@+id/myTextView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#ff0000ff"
    android:padding="3px"
    android:text="Enter User Name"
    android:textSize="16sp"
    android:textStyle="bold"
    android:gravity="center"
    android:layout_x="20px"
    android:layout_y="22px" >
</TextView>
</AbsoluteLayout>
```





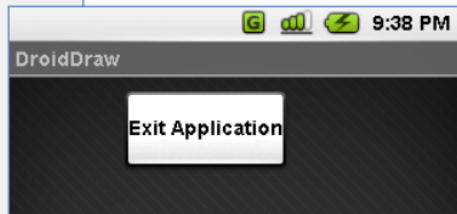
Les classes View : Buttons

- **Button** est utilisé pour simuler une action de click, c'est une sous classe de **TextView**.
- Référence Android Developer : <http://developer.android.com/guide/topics/ui/controls/button.html>



Les classes View : Buttons

```
....  
<Button  
    android:id="@+id/btnExitApp"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:padding="10px"  
    android:layout_marginLeft="5px"  
    android:text="Exit Application"  
    android:textSize="16sp"  
    android:textStyle="bold"  
    android:gravity="center"  
    android:layout_gravity="center_horizontal"  
>  
</Button>
```



```
<?xml version="1.0" encoding="utf-8"?>  
<Button xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/button_send"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_send"  
    android:onClick="sendMessage" />
```

```
/** Called when the user touches the button */  
public void sendMessage(View view) {  
    // Do something in response to button click  
}
```




Les classes View : ImageView & ImageButton

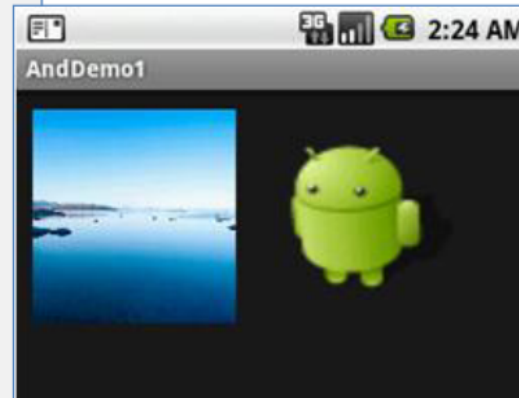
- **ImageView & ImageButton** permettent d'afficher une image via les propriétés `android:src` ou `android:background`.

```

...
<ImageButton
    android:id="@+id/myImageButton1"
    android:background="@drawable/default_wallpaper"
    android:layout_width="125px"
    android:layout_height="131px"
>
</ImageButton>

<ImageView
    android:id="@+id/myImageView1"
    android:background="@drawable/ic_launcher_android"
    android:layout_width="108px"
    android:layout_height="90px"
>
</ImageView>

```





Les classes View : EditText

- **EditText** est un **TextView** éditable (*setText()* & *getText()*).

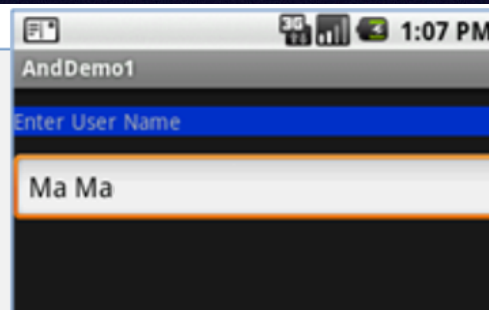
Example

```
...
<EditText
    android:id="@+id/txtUserName"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:autoText="true"
    android:capitalize="words"
    android:hint="First Last Name"
>
</EditText>
...
```

Upper case words

Enter "teh"
will be corrected to: "The"

Suggestion (grey)

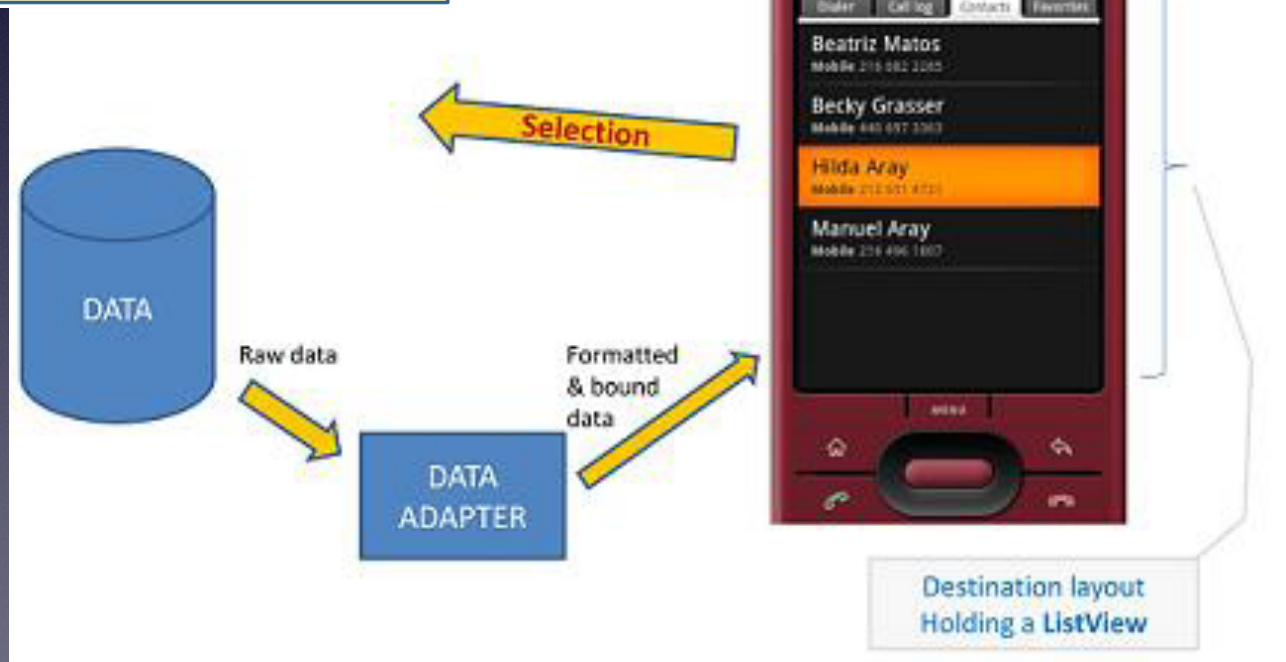


- **android:autoText**
- **android:capitalize**
- **android:digits**
- **android:singleLine**
- **android:password**
- **android:numeric**
- **android:phoneNumber**



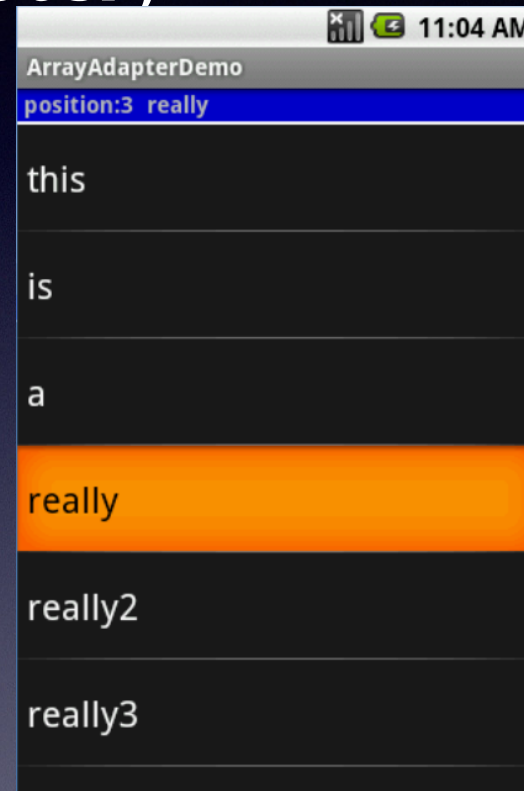
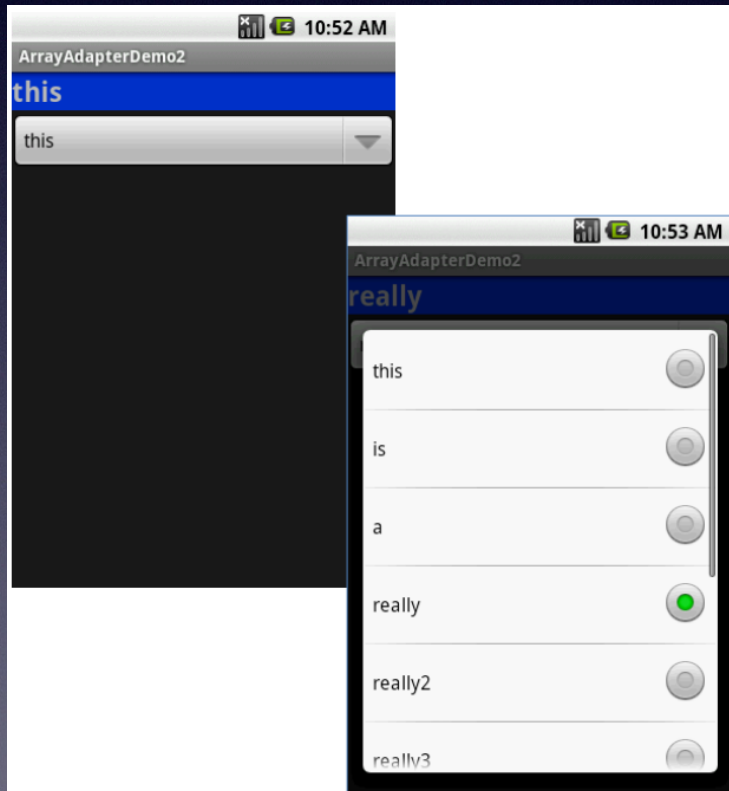
Les classes View : Sélection (*ArrayAdapter*)

```
String[] items={"this", "is", "a","really", "silly", "list"};  
new ArrayAdapter<String>(this,  
    android.R.layout.simple_list_item_1,  
    items);
```





Les classes View : Sélection (*ArrayAdapter*)





Les Notifications

- Informer l'utilisateur de l'application
- Une notification est une indication qui s'affiche sur la barre en haut du téléphone.
- Une notification est cliquable (ouverture de l'application) et effaçable.

Notification shown on the status line

Drag down

Click on Notification Panel to execute associated application

5554: Golfing at Manakiki sat. at 1:00pm

5554: Remember to buy bread and milk
Sent: 8:27PM

5554: Golfing at Manakiki sat. at 1:00pm
Sent: 8:29PM

Type to compose Send



Les Notifications : NotificationManager

- Le **NotificationManager** notifie l'utilisateur des événements qui se passent en tâche de fond.
- Les *Notifications* prennent plusieurs formes:
 - Une icône persistante dans la barre de statut qui permet de lancer un intent et donc une application
 - Le clignotement de la LED du mobile
 - Des alertes complémentaires:
 - en éclairant l'écran
 - Jouant un son
 - Faisant vibrer le mobile
- Le **NotificationManager** est accessible via `getSystemService()`:
`String servName = Context.NOTIFICATION_SERVICE;`
`notificationManager = (NotificationManager) getSystemService (servName);`



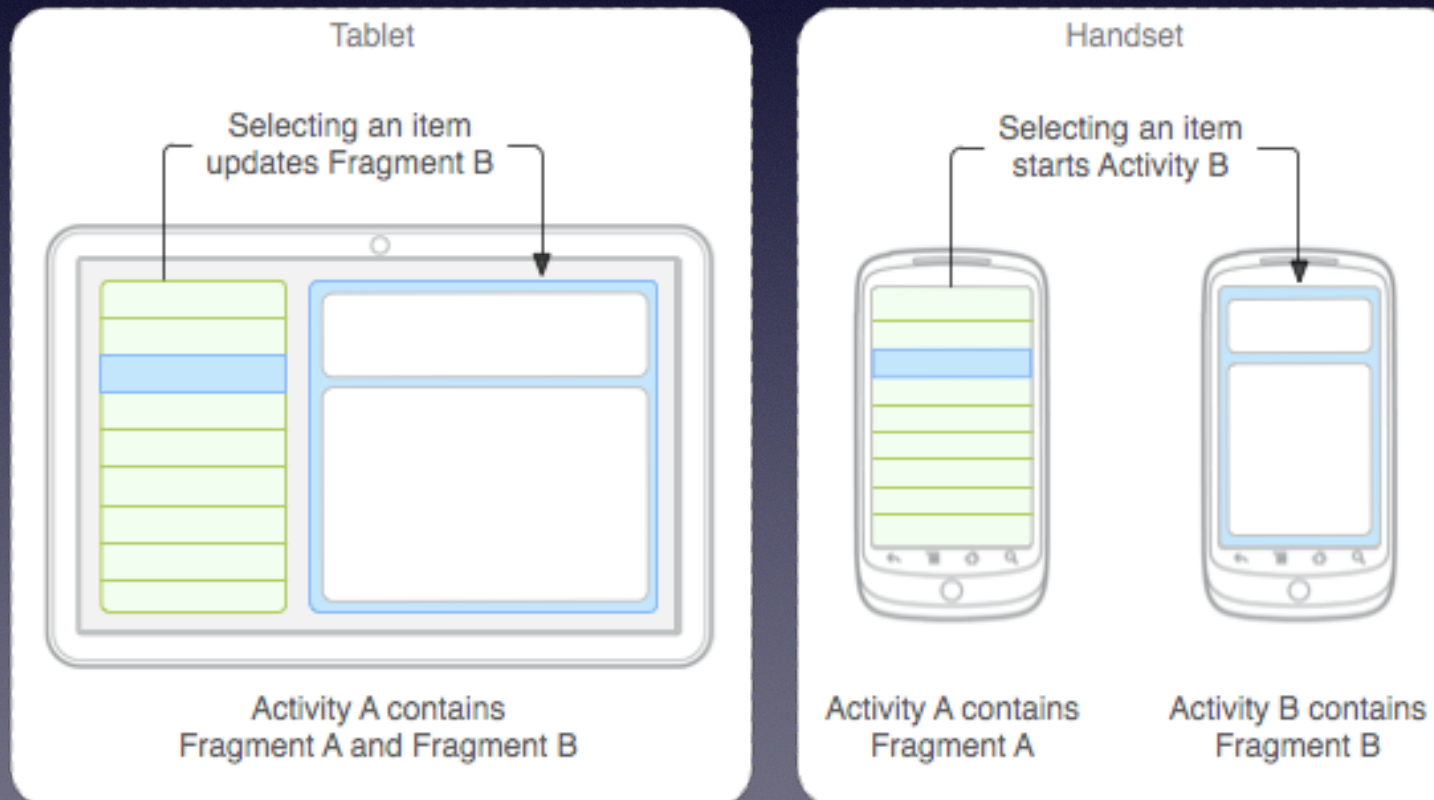
Les Notifications

- La *Notification* représente comment une notification persistante sera présentée via le **NotificationManager**.
- Paramètres (l'identifiant de l'icône, le texte visible dans la barre et le temps d'affichage)
- Méthodes pour gérer la notification:
 - *notify*(int id, Notification notification)
 - *setLatestEventInfo*(Context context, CharSequence contentTitle, CharSequence contentText, PendingIntent contentIntent)
 - *cancel*(int id)
 - *cancelAll*()



Les Fragments

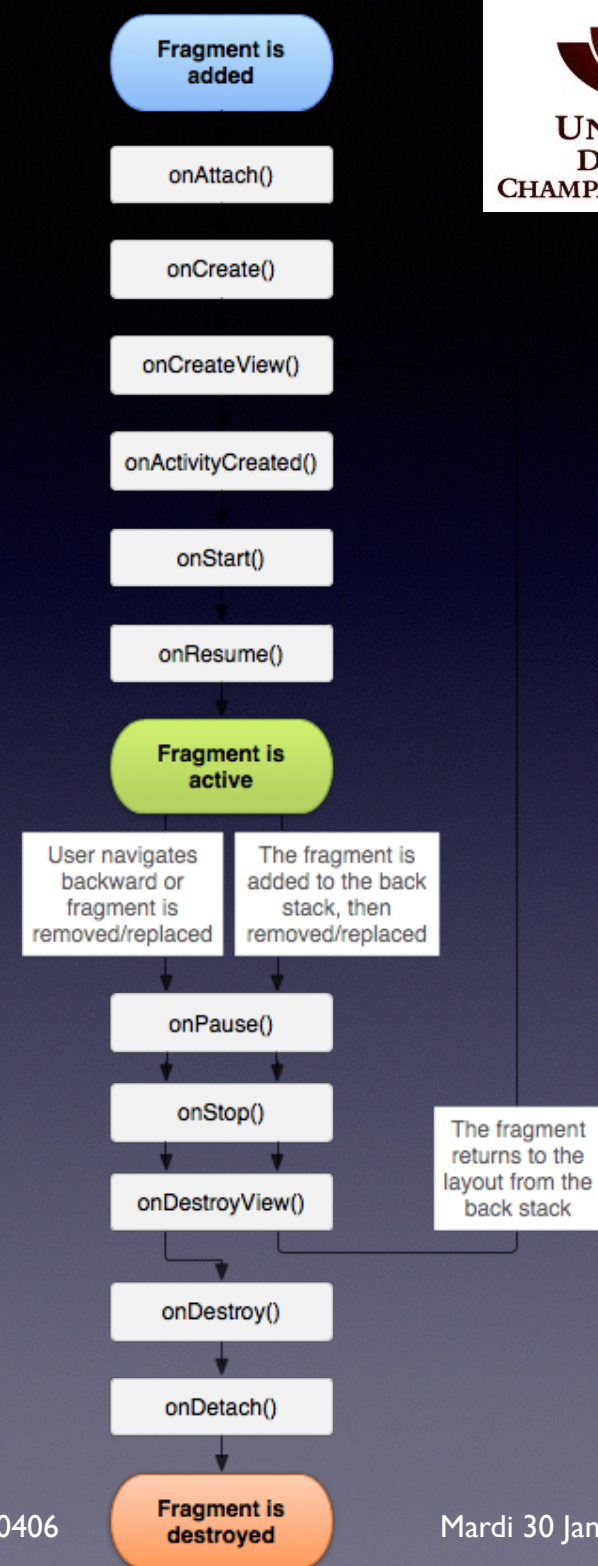
- Optimiser vos applications pour tablettes et mobiles
- Les **Fragments** permettent de définir des interfaces graphiques qui peuvent être réutilisées pour composer de nouvelles interfaces riches.





Les Fragments : cycle de vie

- Les **Fragments** ont une structure de code proche de celle d'une *Activity*. Les **Fragments** ont des méthodes similaires (*onCreate()*, *onStart()*, *onPause()*, et *onStop()*).
- La conversion d'une *Activity* en fragments se fait en agrégeant le *Fragment* à l'*Activity* et en déplaçant le code.





Développement d'Applications sous Android

Interface d'Applications d'Android

Post-it

- Les composants graphiques sont hérités de la classe View
- Il existe plusieurs types de Layout qui permettent de placer les éléments graphiques dans une activité : LinearLayout, FrameLayout, RelativeLayout, TableLayout, Gallery, etc.