Les layouts sous Android

1. ViewGroup

- ViewGroup = container de vues gérant leur agencement
- Création d'un nouveau ViewGroup par héritage
  - Redéfinition de onLayout() nécessaire
  - Création d'une classe interne dérivée de ViewGroup.LayoutParams pour les paramètres d'agencement de chaque vue enfant (facultatif)
- Deux types de *ViewGroup*
  - Destinés à l'agencement statique (*Layout) :
    - Il est conseillé de définir l'arbre de vue par une ressource layout XML
    - Manipulation des enfants possible également à l'exécution :
      - Ajout d'un enfant avec ViewGroup.addView(), suppression avec ViewGroup.removeView()
      - Parcours de la liste des enfants avec ViewGroup.getChildAt() et ViewGroup.getChildCount()
  - Destinés à l'agencement dynamique et basés sur un Adapter (modèle) ; efficaces pour afficher un grand nombre d'éléments obtenus à l'exécution avec recyclage des vues

2. Agencement des composants

2.1. Comment implanter un ViewGroup ?

- Réagencement du ViewGroup nécessaire :
  - lorsqu'une modification est réalisée sur les composants enfants du ViewGroup
    - ajout d'un composant enfant, suppression d'un composant enfant
    - changement de taille d'un composant enfant
  - lorsqu'un paramètre d'agencement du ViewGroup est modifié
  - déclenchement manuel du réagencement avec appel à requestLayout()
- Étapes d'un réagencement :
  1. La mesure : quelles dimensions doit avoir le ViewGroup pour afficher convenablement tous ses enfants ?
  2. Le positionnement : où doit-on placer les enfants du ViewGroup ?

2.2. La mesure

- Première étape du réagencement
- Appel récursif de View.measure(int widthMeasureSpec, int heightMeasureSpec) pour que les vues enfants calculent leurs dimensions souhaitées

Les layouts sous Android

  - Les bits de poids fort de widthMeasureSpec et height sont utilisables pour indiquer s'il s'agit de dimensions préférées, exactes ou maximum (utilisation de la classe View.MeasureSpec)
  - View.measure n'est pas redéfinissable, il faut implanter onMeasure(int, int) qui reçoit les dimensions souhaitées par le parent et déclare les dimensions souhaitées par la vue avec setMeasuredDimension(int, int).
  - Consultation des dimensions fixées avec getMeasuredWidth() et getMeasuredHeight()
- Un parent peut appeler plusieurs fois child.measure() avec différentes hypothèses de dimension pour l'enfant.

Conseils pour l'implantation de onMeasure(int widthMeasureSpec, int heightMeasureSpec) :

- On récupère le type de contrainte communiquée sur la largeur et la hauteur avec MeasureSpec.getMode(dimension) :
  - MeasureSpec.EXACTLY indique que l'on souhaite que le ViewGroup occupe exactement l'espace indiqué
  - MeasureSpec.AT_MOST indique que l'espace indiqué est une valeur maximale à ne pas dépasser
  - MeasureSpec.UNSPECIFIED indique qu'il n'y a pas de contrainte, le ViewGroup peut donc proposer ses dimensions préférées
- Récupérer la valeur encodée de chaque dimension avec MeasureSpec.getSize(dimension)
- Connaissant chaque dimension souhaitée et sa contrainte associée, on peut calculer l'espace nécessaire pour afficher le ViewGroup
  - On tient compte des besoins intrinsèques du ViewGroup (marges, éléments de décors...)
  - On considère également les besoins pour l'affichage des composants enfants du ViewGroup :
    - Pour cela, on interroge chaque enfant sur les dimensions qu'il souhaite avoir
- Lorsque les dimensions du ViewGroup sont calculées, on les fixe avec setMesuredDimension(int width, int height) (appel obligatoire sous peine d'exception levée)
- Si le ViewGroup est contrarié (dimensions mesurées trop faibles pour un affichage correct), on peut appeler setMeasuredDimension(w + View.MEASURED_STATE_TOO_SMALL, h + View.MEASURED_STATE_TOO_SMALL)
- Après appel à setMeasuredDimension, on peut récupérer les valeurs fixées avec getMeasuredWidth() et getMeasuredHeight()
- getMeasuredWidthAndState() & View.MEASURED_STATE_TOO_SMALL != 0 si le drapeau indiquant une longueur trop petite a été fixé (fonctionne similairement pour height)

Les layouts sous Android

Comment interroger les composants enfants sur leurs dimensions souhaitées :

- On appelle child.measure(int widthMeasureSpec, int heightMeasureSpec) pour déclencher l'appel à onMeasure sur l'enfant
- On récupère le résultat avec child.getMeasuredWidth() et child.getMeasuredHeight()
- La mesure peut être renouvelée avec des contraintes différentes, typiquement :
    - D'abord avec MeasureSpec.UNSPECIFIED pour avoir une idée des dimensions préférées des enfants
    - Si l'espace est trop restreint étant données les volontés des enfants, on renouvelle la mesure avec MeasureSpec.AT_MOST
    - Si au contraire il reste de l'espace inoccupé, on peut forcer un enfant à adopter des dimensions avec la contrainte MeasureSpec.EXACTLY

2.3. Le positionnement

- Appel récursif de View.layout(int left, int top, int right, int bottom) pour positionner les vues enfants dans le cadre spécifié (doit être réalisé après la mesure)
- View.layout appelle View.onLayout(boolean changed, int l, int l, int r, int b) qui doit être redéfini
- La vue appelle pour chaque enfant child.layout(...) en s'aidant de child.getMeasured{Width, Height}() calculés lors de l'étape de mesure

3. Les dimensions

Possibilité de définir des dimensions pour les composants ajoutés sur un layout :

- Pour spécifier des dimensions minimales pour le composant avec minWidth et minHeight
- Pour indiquer une marge avec layout_margin (les marges sont également redéfinissables individuellement aux quatre coins avec layout_marginLeft, layout_marginTop, layout_marginRight, layout_marginBottom...)
- Pour spécifier un padding avec padding (également redéfinissable avec les quatre coins)
    - Ne pas confondre *margin* et *padding* : *margin* est une propriété du layout tandis que *padding* est une propriété propre du composant. Par exemple pour un bouton *margin* introduit un espace autour de lui tandis que *padding* réserve de l'espace à l'intérieur du bouton pour séparer le texte du bord du bouton.
- Pour indiquer au layout les dimensions souhaitées pour le composant avec layout_width et layout_height`: on peut utiliser les constantes MATCH_PARENT ou WRAP_CONTENT ou une valeur absolue.

4. Layouts

Les layouts sous Android

L'API propose certains layouts permettant de créer un agencement de composants sous la forme de fichiers XML (ou programmatiquement).

Lorsque l'on place un composant graphique sur un layout, des paramètres de placements (*LayoutParams*) peuvent être utilisés (sous la forme d'attributs XML ou d'une instance de LayoutParams).

Paramètres d'agencement communs à tous les layouts :

- *width* et *height* pour indiquer les dimensions demandées pour le composant enfant
    - Spécifiés à l'aide de valeurs absolues
    - Ou alors avec des constantes spéciales :
        - *MATCH_PARENT* si la dimension doit occuper tout l'espace fourni par le parent
        - *WRAP_CONTENT* si la dimension doit être réduite au minimum nécessaire pour afficher le contenu du composant (par exemple le texte d'un bouton ou d'un TextView)
        - *0dp* si la dimension doit être imposée par des contraintes de placement sur le layout

Approches possibles pour mettre en oeuvre un agencement complexe :

- Créer programmatiquement une nouvelle classe héritée de ViewGroup avec la redéfinition de onMeasure et onLayout
- Imbriquer différents *ViewGroup* déjà fournis par l'API ≤ Approche la plus courante

4.1. FrameLayout

- Affichage d'une pile de vues avec gestion basique d'agencement
- Les vues sont placées les unes sur les autres dans l'ordre de leur ajout (les dernières vues ajoutées peuvent masquer les vues en fond)
- La taille du layout s'adapte aux dimensions du plus grand enfant
- Paramètre d'agencement :
    - *gravity* définit l'emplacement de la vue enfant (différentes constantes combinables par | : top, bottom, left, right, center, fill...) si celle-ci est plus petite que les dimensions du layout

Les layouts sous Android

Exemple de *FrameLayout* :


```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools=http://schemas.android.com/tools
    android:layout_width="match_parent"
    android:id="@+id/testFrameLayout"
    android:layout_height="match_parent"
    tools:context=".layoutest.LayoutestActivity">
    <ImageView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:id="@+id/imageView"
        android:layout_gravity="center"
        android:src="@android:drawable/sym_action_call"/>
    <AnalogClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/analogClock"
        android:layout_gravity="left|top" />
    <Button
```

Les layouts sous Android

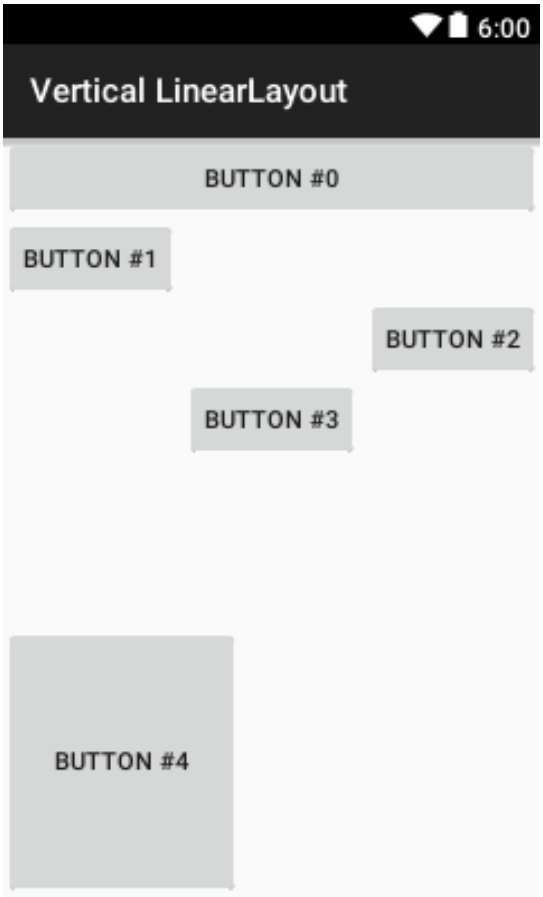
```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Answer the phone"
        android:id="@+id/button1"
        android:layout_gravity="right|bottom"
        android:paddingBottom="100dp" />
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView2"
        android:src="@android:drawable/arrow_down_float"
        android:minHeight="100dp"
        android:minWidth="100dp"
        android:layout_gravity="right|top" />
</FrameLayout>
```

4.2. LinearLayout

- Agencement des enfants dans une direction unique
- Attribut d'orientation : LinearLayout.HORIZONTAL et LinearLayout.VERTICAL
- Paramètres d'agencement :
    - *gravity* pour le placement de composants plus petits que l'espace qui leur sont alloués
    - *weight* pour la priorité pour l'agrandissement ou la réduction d'un composant (distribution de la différence d'espace proportionnelle au poids)
        - Par exemple si l'on place trois boutons en orientation horizontale avec *weights* de 0, 1 et 2, pour la dimension x, chaque bouton occupe la longueur nécessaire pour l'affichage de son texte ; s'il reste de l'espace libre, le 1er bouton n'est pas agrandi, le 2ème bouton capte 1/3 de l'espace supplémentaire, le deuxième bouton 2/3.

Les layouts sous Android

Exemple avec une orientation verticale :




```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/testLinearLayout"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:orientation="vertical">
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button #0"
        android:id="@+id/button0" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button1"
        android:text="Button #1"
        android:layout_gravity="left" />
    <Button
```

Les layouts sous Android

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button2"
        android:text="Button #2"
        android:layout_gravity="right" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button3"
        android:text="Button #3"
        android:layout_gravity="center" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="1dp"
        android:layout_weight="1"
        android:id="@+id/button4"
        android:text="Button #4"
        android:padding="30dp"
        android:layout_marginTop="100dp"
        android:layout_gravity="left" />
</LinearLayout>
```

4.3. RelativeLayout

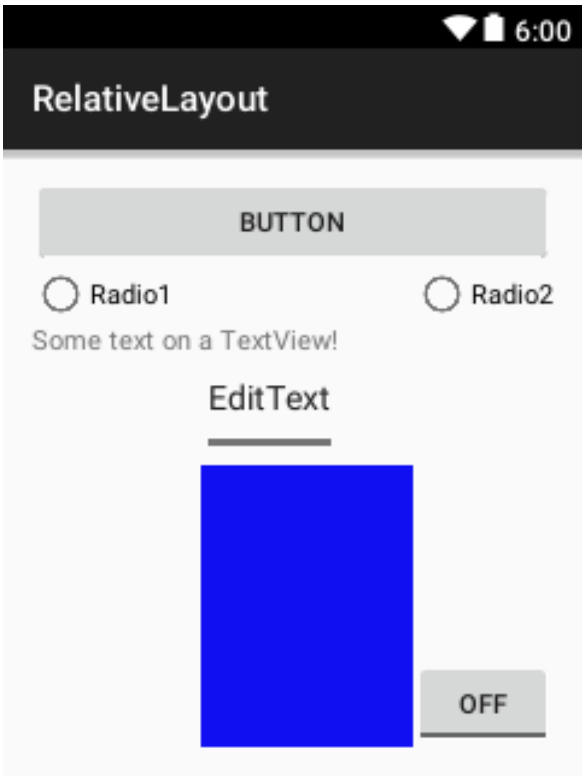
- Permet l'agencement de composants avec la spécification de contraintes relatives au composant parent et aux composants frères (on place tel composant à gauche d'un autre, en occupant toute la hauteur du composant parent...)
- Très pratique pour la réalisation de formulaires (placement de champs d'édition, de boutons...)
- Attention à éviter les contraintes circulaires
- L'utilisation d'un éditeur WYSIWYG d'un IDE peut être utile
- Paramètres d'agencement (on spécifie l'ID désignant l'autre composant) :
    - Paramètres de positionnement relatif :
        - Verticalement : below (en bas d'un composant), above (au-dessus d'un composant)
        - Horizontalement :
            - toLeftOf (à gauche de), toRightOf (à droite de)
            - toStartOf (avant), toEndOf (après)
            - Il faut privilégier start et end plutôt que left et right pour les formulaires pour une adaptation au sens d'écriture (de droite à gauche pour les langues comme l'hébreu, le syriaque, l'arabe...). Il existe aussi des écritures cursives de haut en bas (mongol traditionnel) qui ne sont

Les layouts sous Android

pas supportées nativement par Android. Le boustrophédon (aucun système d'écriture moderne ne l'utilise) n'est pas supporté également.

- Paramètres d'alignement par rapport à un composant frère :
  - Verticalement : alignTop, alignBottom, alignBaseline pour faire coincider le haut ou la bas du composant (ou la ligne de base) avec un autre
  - Horizontalement : alignLeft, alignRight, alignStart, alignEnd
- Paramètres d'alignement par rapport au parent (booléens)
  - Verticalement : alignParentTop, alignParentBottom
  - Horizontalement : alignParentLeft, alightParentRight, alignParentStart, alignParentEnd
  - Centrage : centerHorizontal, centerVertical, centerInParent (centrage horizontal et vertical)

Exemple de *RelativeLayout* :




```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools=http://schemas.android.com/tools
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/testRelativeLayout"
    tools:context=".layoutest.LayoutestActivity">
   <Button
```

Les layouts sous Android

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        android:id="@+id/button"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />
    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Radio1"
        android:id="@+id/radio0"
        android:layout_below="@+id/button"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:checked="false" />
    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Radio2"
        android:id="@+id/radio1"
        android:layout_below="@+id/button"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:checked="false" />
    <ToggleButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New ToggleButton"
        android:id="@+id/toggleButton"
        android:layout_alignParentBottom="true"
        android:layout_alignRight="@+id/radio1"
        android:layout_alignEnd="@+id/radio1"
        android:layout_alignLeft="@+id/radio1"
        android:layout_alignStart="@+id/radio1"
        android:checked="false" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Some text on a TextView!"
        android:id="@+id/textView3"
        android:layout_below="@+id/radio0"
        android:layout_alignParentLeft="true"
```

Les layouts sous Android

```
        android:layout_alignParentStart="true" />
    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText"
        android:layout_below="@+id/textView3"
        android:layout_alignRight="@+id/textView3"
        android:layout_alignEnd="@+id/textView3"
        android:text="EditText" />
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView"
        android:layout_below="@+id/editText"
        android:layout_alignLeft="@+id/editText"
        android:layout_alignStart="@+id/editText"
        android:layout_alignBottom="@+id/toggleButton"
        android:layout_toLeftOf="@+id/toggleButton"
        android:layout_toStartOf="@+id/toggleButton"
        android:src="#0f0ff1" />
</RelativeLayout>
```

4.4. TableLayout

- Contient des lignes de type TableRow (qui sont des sortes de LinearLayout)
- Philosophiquement similaire aux système de table en HTML (table, row, td)
- Paramètres d'agencement de TableRow :
    - *column* pour indiquer l'indice de départ de colonne
    - *span* pour indiquer le nombre de colonnes occupées

Les layouts sous Android

Exemple de *TableLayout* :


```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools=http://schemas.android.com/tools
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/testTableLayout"
    tools:context=".layoutest.LayoutestActivity">
    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="7"
            android:id="@+id/button7"
            android:layout_column="0" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="8"
            android:id="@+id/button8"
            android:layout_column="1" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

Les layouts sous Android

```
            android:text="9"
            android:id="@+id/button9"
            android:layout_column="2"
            android:layout_weight="1" />
    </TableRow>
    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="4"
            android:id="@+id/button4"
            android:layout_column="0"
            android:layout_weight="0" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="5"
            android:id="@+id/button5"
            android:layout_column="1"
            android:layout_weight="1" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="6"
            android:id="@+id/button6"
            android:layout_column="2"
            android:layout_weight="0" />
    </TableRow>
    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="1"
            android:id="@+id/button1"
            android:layout_column="0"
            android:layout_weight="1" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="2"
            android:id="@+id/button2"
```

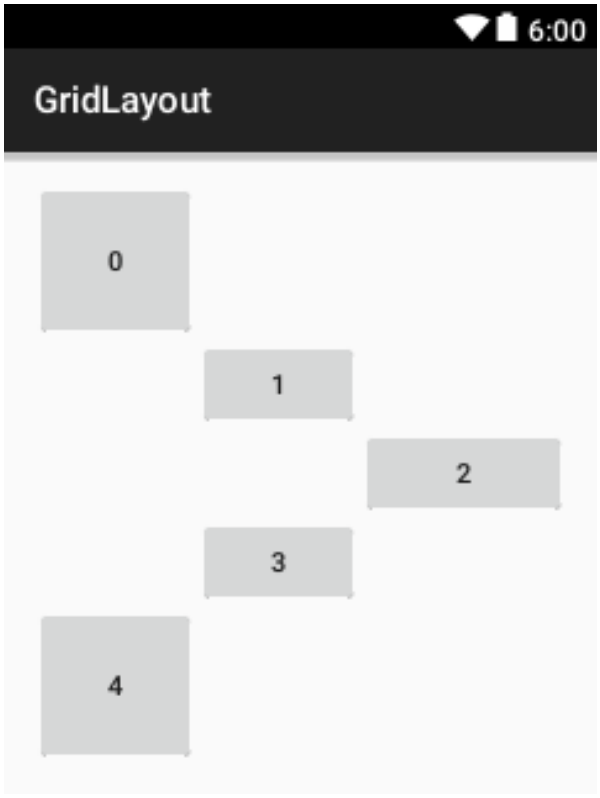
Les layouts sous Android

```
            android:layout_column="1"
            android:layout_weight="0" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="3"
            android:id="@+id/button3"
            android:layout_column="2"
            android:layout_weight="1" />
    </TableRow>
    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="0"
            android:id="@+id/button0"
            android:layout_column="0"
            android:layout_span="3"
            android:layout_weight="1"
            android:paddingTop="100dp"
            android:focusableInTouchMode="false"
            android:textSize="50sp" />
    </TableRow>
</TableLayout>
```

4.5. GridLayout

- Agencement sur une grille rectangulaire de N colonnes
    - contrairement au TableLayout, les composants sont ajoutés directement avec leur paramètres d'agencement
- Propriétés de GridLayout.LayoutParams :
    - *column* et *columnSpan* : colonne de départ et nombre de colonnes occupées
    - *row* et *rowSpan* : ligne de départ et nombre de lignes occupées
    - *gravity* : emplacement de la vue enfant dans la cellule

Les layouts sous Android




```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:id="@+id/testGridLayout"
    android:layout_height="match_parent"
    android:columnCount="3"
    android:rowCount="5">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="0"
        android:id="@+id/button0"
        android:layout_row="0"
        android:layout_column="0"
        android:layout_rowWeight="1"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="1"
        android:id="@+id/button1"
        android:layout_row="1"
        android:layout_column="1" />
    <Button
```

Les layouts sous Android

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="2"
        android:id="@+id/button2"
        android:layout_row="2"
        android:layout_column="2"
        android:layout_columnWeight="1"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="3"
        android:id="@+id/button3"
        android:layout_row="3"
        android:layout_column="1" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="4"
        android:id="@+id/button4"
        android:layout_row="4"
        android:layout_column="0"
        android:layout_rowWeight="1"/>
</GridLayout>
```

4.6. ConstraintLayout

- Layout permettant de définir des contraintes de placement : comparable au RelativeLayout mais avec plus de possibilités
- A pour vocation d'être un layout universel répondant à la plupart des besoins : usage par défaut lors de la création d'une activité Android avec Android Studio
- Layout en développement actif : il est conseillé d'utiliser une bibliothèque de rétrocompatibilité pour utiliser des fonctionnalités récentes sur des versions anciennes de l'API
- Différents types de contraintes disponibles (cf Javadoc)

4.6.1. Contraintes de placement relatif

Positionnement relatif possible par rapport à un autre composant (en indiquant son id) ou par rapport au parent (désigné par *parent*).

- Sur l'axe horizontal pour aligner un côté d'un composant avec le côté d'un autre composant : layout_constraint{Left, Right, Start, End}_to{Left, Right, Start, End}Of
- Sur l'axe vertical par rapport au bas, haut ou ligne de base (alignement de la base des lettres) des composants : layout_constraint{Top, Bottom, Baseline}_to{Top, Bottom, Baseline}Of

Les layouts sous Android

- Sur un repère circulaire pour placer le centre d'un composant par rapport au centre d'un autre composant en indiquant le composant de référence au centre du cercle (layout_constraintCircle), le rayon du cercle (layout_constraintCircleRadius) et l'angle de placement (layout_constraintCircleAngle)

4.6.2. Dimensions

4.6.2.1. Marges

Possibilité de définir des marges (comme pour la plupart des layouts) :

- layout_margin{Left, Right, Start, End, Top, Bottom} par rapport à un composant visible

Si un composant disparaît en visibilité GONE, la marge du composant adjacent continue d'être appliquée par rapport au prochain composant visible. Ce comportement est modifiable en indiquant une marge spéciale si le composant adjacent est GONE :

- layout_goneMargin{Left, Right, Start, End, Top, Bottom} par rapport à un composant de visibilité GONE

4.6.2.2. Largeur et hauteur

Possibilité de définir comme pour tous les layouts les dimensions d'un composant enfant (layout_width et layout_height) en utilisant les valeurs suivantes :

- WRAP_CONTENT : dimension préférée pour l'affichage du composant
- MATCH_PARENT : utilisation de tout l'espace du parent (ConstraintLayout) ; à proscrire (utiliser plutôt des contraintes de placement relatives au parent)
- MATCH_CONSTRAINT : pour laisser les contraintes de placement déterminer la taille
    - Possibilité de définir une taille en ratio par rapport au parent avec layout_constraint{Width, Height}_percent (valeur flottante entre 0.0 et 1.0)
    - Possibilité de définir un ratio de la largeur du composant par rapport à sa hauteur avec layout_constraintDimensionRatio :
        - avec une valeur flottante (0.5 par exemple si la largeur est deux fois plus petite que la hauteur)
        - avec une valeur fractionnaire (16:9 par exemple)
- Une dimension "en dur" (en px, dp...)
    - la valeur indiquée est "minimale", la dimension peut être agrandie si les contraintes l'autorisent
    - ainsi 0dp équivaut à MATCH_CONSTRAINT

4.6.2.3. Contraintes contradictoires

- Possibilité de définir plusieurs contraintes de placement relatif
    - ContraintLayout essaie de satisfaire toutes les contraintes...
    - ...mais il peut y avoir des contraintes contradictoires notamment avec les dimensions

Les layouts sous Android

Exemple : définissons un TextView avec les contraintes suivantes :

- Placement sur la gauche du parent
- Placement sur la droite du parent
- Largeur en WRAP_CONTENT

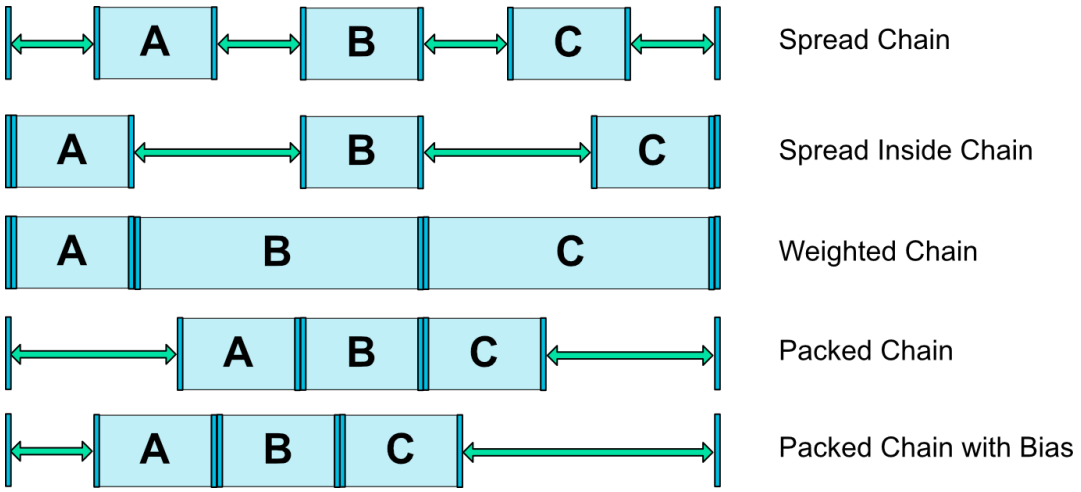
Si le texte du TextView est plus petit que la largeur du parent, les trois conditions sont impossibles à satisfaire simultanément.

Solution proposée par ConstraintLayout :

- par défaut centrer le composant horizontalement
- le comportement est modifiable en spécifiant des *biais* : layout_constraintHorizontal_bias="0.25" permet de rapprocher le composant du bord gauche (25% d'espace à gauche, 75% à droite)

4.6.3. Chaînes

- Groupement possible des composants avec des chaînes sur un axe horizontal ou vertical
- Différents types de chaînes (cf image suivantes tirée de la Javadoc)



- Pour les weighted chains, il est possible de définir un poids pour chaque composant avec l'attribut layout_constraint{Horizontal, Vertical}_bias
- Un composant peut participer à la fois à une chaîne horizontale et une chaîne verticale

4.6.4. Flow : des chaînes avec retour à la ligne

- Composant helper pour *ConstraintLayout* permettant de créer des chaînes d'éléments autorisant le retour à la ligne
- Un peu analogue aux boîtes *flex* en CSS
- Créer un composant *flow* :
    1. Création depuis le menu contextuel de l'UI layout d'Android Studio (*Helpers>Add flow*)
    2. Ajout des références de composants du flow par glisser-déposer

3. Configuration des propriétés du *flow* :
    - flow_wrapMode
        - none : pas de retour à la ligne ou à la colonne (comportement d'une chaîne standard)
        - chain : création de chaînes multiples sur plusieurs lignes ou colonnes
        - align : idem que chain mais avec un alignement des items (comme pour un GridLayout)
    - flow_horizontalStyle et flow_verticalStyle : pour configurer le regroupement des items sur une chaîne
        - spread : répartition avec éloignement des bords
        - spread_inside : répartition avec alignement du premier et dernier item sur les bords
        - packed : regroupement au milieu

- Page de référence de la Javadoc

4.6.5. Barrières et guides

- Possibilité de placer deux types de repères horizontaux ou verticaux servant de références d'alignement
    - *Barrier* : le repère est placé le plus à {gauche, droite, haut, bas} d'un groupe de composants
        - Utile pour des formulaires pour aligner des champs
    - *Guideline* : le repère est placé horizontalement ou verticalement à un ratio fixé de la largeur/hauteur
- Des contraintes peuvent ensuite être exprimées par rapport au repère (à droite de la barrière, sous le guide...)

4.7. Groupes

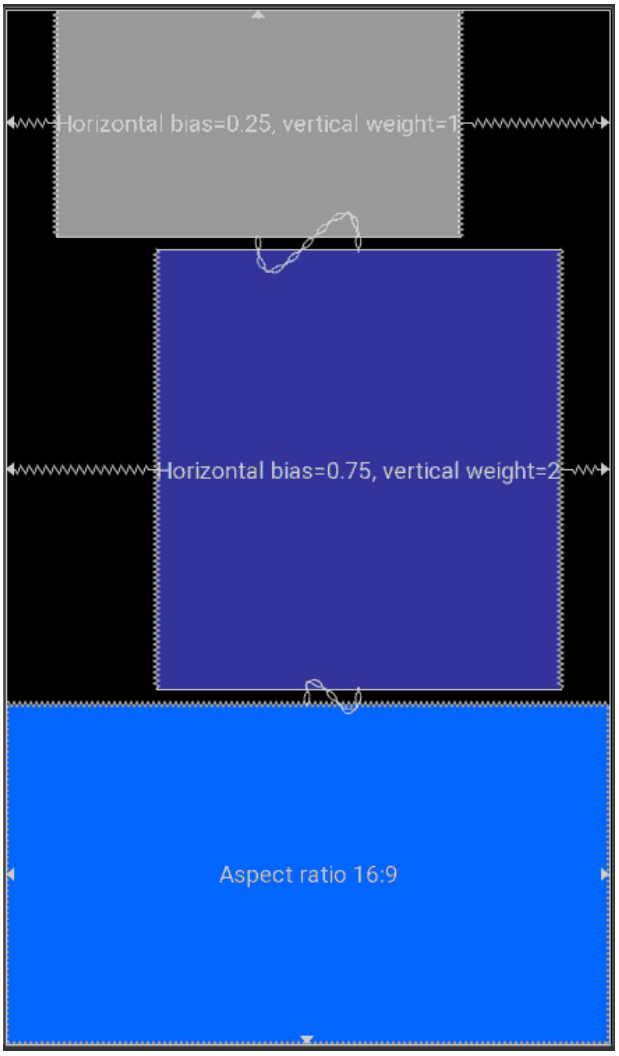
- Un groupe permet de définir un ensemble de composants sur lesquels on peut appliquer la même politique de visibilité (masque ou affiche les composants avec une seule instruction)
- Un composant peut être inclus dans plusieurs groupe ; c'est la visibilité du dernier groupe déclaré qui impose la visibilité du composant

Exemple avec un groupe référençant quatre boutons par leur id :

```
<androidx.constraintlayout.widget.Group
                android:id="@+id/group"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:visibility="visible"
                app:constraint_referenced_ids="button0,button1,button2,button3"
/>
```

Les layouts sous Android

4.7.1. Exemple de ConstraintLayout




```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:background="@android:color/darker_gray"
        android:gravity="center"
        android:text="Horizontal bias=0.25, vertical weight=1"
        app:layout_constraintBottom_toTopOf="@id/textView2"
```

Les layouts sous Android

```
        app:layout_constraintHorizontal_bias="0.25"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_chainStyle="spread"
        app:layout_constraintVertical_weight="1.0" />
    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_marginTop="8dp"
        android:background="@color/colorPrimaryDark"
        android:gravity="center"
        android:text="Horizontal bias=0.75, vertical weight=2"
        app:layout_constraintBottom_toTopOf="@id/textView3"
        app:layout_constraintHorizontal_bias="0.75"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/textView1"
        app:layout_constraintVertical_weight="2.0" />
    <TextView
        android:id="@+id/textView3"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginTop="8dp"
        android:background="#0f50dd"
        android:gravity="center"
        android:text="Aspect ratio 16:9"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintDimensionRatio="16:9"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/textView2" />
    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        app:layout_constraintGuide_begin="20dp" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

5. Défilement

Comment rendre un layout défilable ?

Les layouts sous Android

- En l'embarquant comme unique enfant d'un ScrollView : le layout est alors défilable verticalement
- HorizontalScrollView sert à rendre défilable un layout horizontalement
- L'usage d'un NestedScrollView est recommandé pour imbriquer une zone défilable dans une autre zone défilable

Attention, certains composants graphiques intègrent déjà par défaut le défilement vertical (inutile dès les insérer dans un ScrollView), il s'agit de :

- ListView
- GridView

Source : M. Chilowicz (and other specified contributors)