

Programmation mobile

Cours 3 : [View](#)

Julien Grange <julien.grange@lacl.fr>

Mardi 8 octobre 2024

Rappel : `View` et `ViewGroup`

- Le layout d'une `Activity` est constitué d'un arbre de `View`
- Les nœuds internes héritent de `ViewGroup`, qui hérite de `View`, e.g.
 - `ConstraintLayout`
 - `LinearLayout`
 - `Spinner`
- les feuilles héritent directement de `View`, e.g.
 - `TextView`
 - `EditView`
 - `Button`
 - `ImageView`

- 1 `ConstraintLayout` : comment ajouter dynamiquement des `View` et en ajuster les contraintes
- 2 Animer ses `View` avec `ObjectAnimator`
- 3 Comment créer ses propres `View`, avec `onDraw()` et `onTouchEvent()`

- Évite les imbrications de `ViewGroup` → réduit la profondeur de l'arbre.
- Ses enfants sont liés via des *contraintes* :
 - chaînes horizontales/verticales
 - alignements

ConstraintLayout

- Évite les imbrications de `ViewGroup` → réduit la profondeur de l'arbre.
- Ses enfants sont liés via des *contraintes* :
 - chaînes horizontales/verticales
 - alignements
- Pour l'instant, on a vu comment gérer ces contraintes dans le XML

The screenshot shows an IDE with the following components:

- Code Editor:** Displays XML code for three `ConstraintLayout` instances. Each instance contains a `TextView` and a `Button` with various constraints like `start_toStart`, `end_toEnd`, `widthMatchParent`, and `heightMatchParent`.
- Visual Diagram:** A central diagram showing a vertical chain of three `TextView` boxes, with a `Button` box positioned below the middle `TextView`. Dashed lines indicate the constraints between these elements.
- Properties Panel:** On the right, the properties for a `TextView` are shown, including `id`, `layout_width`, `layout_height`, `layout_constraintStart_toStartOf`, `layout_constraintEnd_toEndOf`, `layout_constraintTop_toTopOf`, `layout_constraintBottom_toBottomOf`, `text`, `id`, `inputType`, and `numberDecimal`.
- Constraints Panel:** Below the properties, a list of constraints is shown, including `layout_width` and `layout_height` set to `wrap_content`.

Ajouter dynamiquement une `View`

La présence d'une `View` peut dépendre de l'exécution de l'application. Il faut alors l'ajouter dynamiquement (i.e. à runtime) :

```
ConstraintLayout layout = findViewById(R.id.constraint_layout);

Button button = new Button(this);
button.setText("foo");
button.setOnClickListener(...);

layout.addView(button);
```

Ajouter dynamiquement une `View`

La présence d'une `View` peut dépendre de l'exécution de l'application. Il faut alors l'ajouter dynamiquement (i.e. à runtime) :

```
ConstraintLayout layout = findViewById(R.id.constraint_layout);

Button button = new Button(this);
button.setText("foo");
button.setOnClickListener(...);

layout.addView(button);
```

Pour pouvoir ajuster les contraintes liées à une `View`, il lui faut un id. L'attribution est automatique durant l'expansion du XML ; ici, il faut le faire à la main.

Pour être sûr d'éviter les collisions, on utilise `View.generateViewId()` :

```
button.setId(View.generateViewId());
```

Gérer les contraintes : `ConstraintSet`

Trois étapes :

- 1 Créer un `ConstraintSet` :

```
ConstraintSet constraintSet = new ConstraintSet();  
constraintSet.clone(constraintLayout);
```

- 2 Ajouter des contraintes :

```
constraintSet.connect(...);  
constraintSet.createHorizontalChain(...);
```

- 3 Appliquer ce `ConstraintSet` :

```
constraintSet.applyTo(constraintLayout);
```

- Aligner le bord gauche de view1 et view2 :

```
constraintSet.connect(  
    view1.getId(), ConstraintSet.LEFT,  
    view2.getId(), ConstraintSet.LEFT,  
    0);
```

- Aligner le bord gauche de view1 et view2 :

```
constraintSet.connect(  
    view1.getId(), ConstraintSet.LEFT,  
    view2.getId(), ConstraintSet.LEFT,  
    0);
```

- Centrer verticalement view :

```
constraintSet.connect(  
    view.getId(), ConstraintSet.TOP,  
    constraintLayout.getId(), ConstraintSet.TOP,  
    0);  
constraintSet.connect(  
    view.getId(), ConstraintSet.BOTTOM,  
    constraintLayout.getId(), ConstraintSet.BOTTOM,  
    0);
```

- Créer une chaîne horizontale contenant view1, view2 et view3 :

```
int[] idArray = {view1.getId(),view2.getId(),view3.getId()};  
float[] weightArray = {1f,1f,1f};  
  
constraintSet.createHorizontalChain(  
    constraintLayout.getId(), ConstraintSet.LEFT,  
    constraintLayout.getId(), ConstraintSet.RIGHT,  
    idArray, weightArray,  
    ConstraintSet.CHAIN_SPREAD);
```

MainActivity layout

The screenshot displays the Android Studio IDE with the following components:

- Menu Bar:** File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help.
- Toolbar:** Includes icons for app, Pixel XL API 30, and various development tools.
- Project View:** Shows the project structure with files like `activity_main.xml`, `MainActivity.java`, `LigneBrisee.java`, and `Sommet.java`.
- Palette:** Lists widget categories such as Common, Text, Buttons, Widgets, Layouts, Containers, Helpers, Google, Legacy, and Project.
- Component Tree:** Shows the hierarchy: `constraintLayout` containing `imageView`.
- Design View:** A dark teal background with a white wavy shape and a white rectangular area labeled "imageView".
- Attributes Panel:** Shows properties for `constraintLayout`, including:
 - Declared Attributes:** `layout_width` (match_parent), `layout_height` (match_parent), `context` (.MainActivity), `id` (constraintLayout).
 - Layout:** `layout_width` (match_parent), `layout_height` (match_parent), `visibility` (visible).
 - Transforms:** `minWidth`, `maxWidth`, `minHeight`, `maxHeight`, `alpha`.
 - All Attributes:** `accessibilityLiveRegion`, `actionBarNewMode`, `addStatesFromChildren` (checked), `alpha`, `alwaysDrawnWithCache` (checked), `animateLayoutChanges` (checked), `animationCache` (checked), `background` (white), `barrierMargin`, `circularflow_angles`, `circularflow_defaultAngle`.
- Status Bar:** Shows "Launch succeeded [2 minutes ago]" and system information: 32:42, LF, UTF-8, 4 spaces.

Ajout de Views et de leurs contraintes

The image shows the Android Studio IDE with the following components:

- Code Editor:** Displays the `onCreate` method of `MainActivity.java`. The code includes:
 - Initialization of `buttonX` and `buttonY` with text "Vers la droite" and "Vers le bas".
 - Setting background colors: `buttonX.setBackgroundColor(Color.RED)` and `buttonY.setBackgroundColor(Color.BLUE)`.
 - Creation of a `LigneBrisee` view.
 - Adding views to the layout: `constraintLayout.addView(buttonX)` and `constraintLayout.addView(buttonY)`.
 - Generating view IDs for `buttonX`, `buttonY`, and `ligneBrisee`.
 - Creating a `ConstraintSet` and cloning it from the `constraintLayout`.
 - Setting up a horizontal chain between `buttonX` and `buttonY` with weights of 40% and 60%.
 - Connecting constraints for `buttonX` and `buttonY` to the `constraintLayout` with margins.
 - Applying the `ConstraintSet` to the layout.
- Emulator:** Shows a mobile device screen with a purple header "Cours3", a portrait of Albert Einstein, and two buttons: a red one labeled "VERS LA DROITE" and a blue one labeled "VERS LE BAS".
- IDE Interface:** Includes a menu bar (File, Edit, View, etc.), a toolbar, a project view on the left, and a status bar at the bottom showing "Launch succeeded (moments ago)".

Turing prend le large

The image shows the Android Studio IDE with the following components:

- Code Editor:** Displays the `MainActivity.java` file. The code includes:
 - Layout constraints for a chain of views.
 - Animation logic for a portrait of Alan Turing, where the x-axis position is controlled by a button click.
 - Two buttons: "TURING CROIT" (red) and "TURING BAVE" (blue).
- Mobile Emulator:** Shows a smartphone screen with the title "Cours3". It displays a portrait of Alan Turing with the two buttons mentioned in the code.
- IDE Interface:** Includes the Project, Resource Manager, and Structure toolbars on the left, and the Run, Logcat, and Terminal panels at the bottom.

Turing prend le large



Dans le `onClick()` de boutonX :

```
float x1 = turing.getX();
ObjectAnimator animator = ObjectAnimator.ofFloat(turing,
    "translationX", x1, x1+200f,x1+50f,x1+125f,x1+100f);
animator.setDuration(2000);
animator.start();
```

On veut être capable de

- tracer un trait qui suit le mouvement sur l'écran
- dessiner un point là où le doigt a été posé ou levé

On veut être capable de

- tracer un trait qui suit le mouvement sur l'écran
- dessiner un point là où le doigt a été posé ou levé

Pour cela, on va créer une `View` maison : `LigneBrisee`

Celle-ci va

- “écouter” les mouvements du doigt via `View::onTouchEvent()`
- stocker les points de passage du doigt dans un tableau de `Sommet`
- tracer des lignes entre ces `Sommet` via `View::onDraw()`
- dessiner un point sur les `Sommet` où le toucher est discontinu

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

Cours3 > app > src > main > java > com > example > cours3 > Sommet

activity_main.xml x MainActivity.java x LigneBrisee.java x Sommet.java x

```
1 package com.example.cours3;
2
3 public class Sommet {
4     private final float x,y;
5
6     public Sommet(float x, float y){
7         this.x = x;
8         this.y = y;
9     }
10
11     public float getX(){ return x;}
12     public float getY(){ return y;}
13
14 }
15
```

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

Cours3 > app > src > main > java > com > example > cours3 > LigneBrisee > onDraw

app

Project activity_main.xml MainActivity.java LigneBrisee.java Sommet.java

```
1 package com.example.cours3;
2
3 import ...
4
11
12 public class LigneBrisee extends View {
13
14     private final static int default_color = Color.BLACK;
15     private Paint paint = new Paint();
16     // liste des sommets
17     private ArrayList<Sommet> sommets = new ArrayList<Sommet>();
18     // listes des sommets où le doigt a été levé
19     private ArrayList<Sommet> sommetsSpeciaux = new ArrayList<Sommet>();
20
21     public LigneBrisee(Context context, int color) {
22         super(context);
23         paint.setColor(color);
24     }
25
26     public LigneBrisee(Context context) {
27         this(context, default_color);
28     }
29
30     // onDraw est appelée à chaque fois que la vue est dessinée
31     @Override
32     protected void onDraw(Canvas canvas) {...}
33
34
44
45     // callback appelé à chaque mouvement de l'utilisateur
46     @Override
47     public boolean onTouchEvent(MotionEvent event) {...}
48
49 }
```

```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Cours3 app src main java com example cours3 LigneBrisee onDraw
activity_main.xml MainActivity.java LigneBrisee.java Sommet.java
30 // onDraw est appelée à chaque fois que la vue est dessinée
31 @Override
32 protected void onDraw(Canvas canvas) {
33     super.onDraw(canvas);
34     for (int i = 0; i < sommets.size()-1; i++) {
35         Sommet source = sommets.get(i);
36         Sommet cible = sommets.get(i+1);
37         canvas.drawLine(source.getText(), source.getY(), cible.getText(), cible.getY(), paint);
38     }
39     for (int i = 0; i < sommetsSpeciaux.size()-1; i++) {
40         Sommet source = sommetsSpeciaux.get(i);
41         canvas.drawCircle(source.getText(), source.getY(), radius, pf, paint);
42     }
43 }
44
45 // callback appelé à chaque mouvement de l'utilisateur
46 @Override
47 public boolean onTouchEvent(MotionEvent event) {
48     super.onTouchEvent(event);
49     Sommet sommet;
50     switch(event.getAction()) {
51         // cas où il y a une discontinuité
52         case MotionEvent.ACTION_DOWN:
53         case MotionEvent.ACTION_UP:
54             sommet = new Sommet(event.getText(), event.getY());
55             sommets.add(sommet);
56             sommetsSpeciaux.add(sommet);
57             break;
58         // cas où un mouvement est détecté
59         // on récupère alors le nombre de points intermédiaires via getHistorySize()
60         case MotionEvent.ACTION_MOVE:
61             for(int i = 0; i < event.getHistorySize(); i++) {
62                 sommet = new Sommet(event.getHistoricalX(i), event.getHistoricalY(i));
63                 sommets.add(sommet);
64             }
65             break;
66         default:
67             // on retrace le View -> appel à ondraw()
68             invalidate();
69             return true;
70     }
71 }
72
73 TODO Problems Terminal Logcat Build Profiler Run AppInspection
Launch succeeded [a minute ago] Event Log Layout Inspector 34:29 LF UTF-8 4 spaces
```

View::onDraw()

- Méthode appelée à chaque fois qu'une `View` se dessine.
- Prend un argument : un `Canvas`

View::onDraw()

- Méthode appelée à chaque fois qu'une `View` se dessine.
- Prend un argument : un `Canvas`
- Pour peindre, il faut une toile (`Canvas`) et un pinceau (`Paint`)

<code>Paint</code>	<code>Canvas</code>
<code>setColor()</code>	<code>drawLine()</code>
<code>setStrokeWidth()</code>	<code>drawCircle()</code>

View::onDraw()

- Méthode appelée à chaque fois qu'une `View` se dessine.
- Prend un argument : un `Canvas`
- Pour peindre, il faut une toile (`Canvas`) et un pinceau (`Paint`)

<code>Paint</code>	<code>Canvas</code>
<code>setColor()</code>	<code>drawLine()</code>
<code>setStrokeWidth()</code>	<code>drawCircle()</code>

```
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    for (int i = 0; i < sommets.size()-1; i++) {
        Sommet source = sommets.get(i);
        Sommet cible = sommets.get(i+1);
        canvas.drawLine(source.getX(), source.getY(),
            cible.getX(), cible.getY(), paint);
    }
    for (int i = 0; i < sommetsSpeciaux.size()-1; i++) {
        Sommet sommet = sommetsSpeciaux.get(i);
        canvas.drawCircle(sommet.getX(), sommet.getY(), 4f, paint);
    }
}
```

View::onTouchEvent()

Callback appelé dès qu'un mouvement est détecté.

View::onTouchEvent()

Callback appelé dès qu'un mouvement est détecté.

Un `MotionEvent` lui est passé en argument :

`MotionEvent::getAction()` permet de savoir s'il s'agit d'un

- `MotionEvent.ACTION_DOWN` : un doigt se pose sur l'écran
On peut récupérer sa position via `MotionEvent::getX()` et `MotionEvent::getY()`

View::onTouchEvent()

Callback appelé dès qu'un mouvement est détecté.

Un `MotionEvent` lui est passé en argument :

`MotionEvent::getAction()` permet de savoir s'il s'agit d'un

- `MotionEvent.ACTION_DOWN` : un doigt se pose sur l'écran
On peut récupérer sa position via `MotionEvent::getX()` et `MotionEvent::getY()`
- `MotionEvent.ACTION_UP` : idem, mais le doigt a quitté l'écran

View::onTouchEvent()

Callback appelé dès qu'un mouvement est détecté.

Un `MotionEvent` lui est passé en argument :

`MotionEvent::getAction()` permet de savoir s'il s'agit d'un

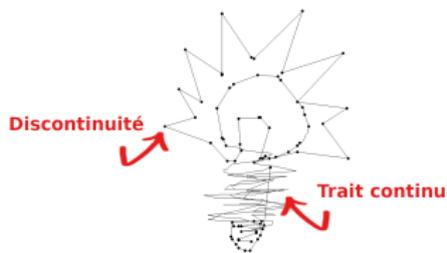
- `MotionEvent.ACTION_DOWN` : un doigt se pose sur l'écran
On peut récupérer sa position via `MotionEvent::getX()` et `MotionEvent::getY()`
- `MotionEvent.ACTION_UP` : idem, mais le doigt a quitté l'écran
- `MotionEvent.ACTION_MOVE` : détection d'un mouvement.
Il s'agit d'une suite de points. On peut récupérer
 - leur nombre via `MotionEvent::getHistorySize()`
 - leurs abscisses via `MotionEvent::getHistoricalX()`
 - leurs ordonnées via `MotionEvent::getHistoricalY()`

View::onTouchEvent()

```
public boolean onTouchEvent(MotionEvent event) {
    super.onTouchEvent(event);
    Sommet sommet;

    switch(event.getAction()) {
        case MotionEvent.ACTION_DOWN:
        case MotionEvent.ACTION_UP:
            sommet = new Sommet(event.getX(), event.getY());
            sommets.add(sommet);
            sommetsSpeciaux.add(sommet);
            break;
        case MotionEvent.ACTION_MOVE:
            for(int i = 0; i<event.getHistorySize(); i++) {
                sommet = new Sommet(event.getHistoricalX(i),
                    event.getHistoricalY(i));
                sommets.add(sommet);
            }
            break;
        default:
    }
    invalidate();           // Redessine la View -> appelle onDraw()
    return true;
}
```

Cours3



VERS LA DROITE

VERS LE BAS

