

Programmation mobile

Cours 3 : [View](#)

Julien Grange <julien.grange@lacl.fr>

Mardi 7 octobre 2025

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`

Au programme du jour...

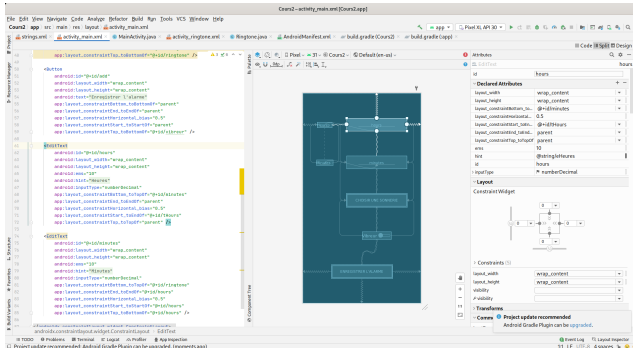
- 1 `ConstraintLayout` : comment ajouter dynamiquement des `View` et en ajuster les contraintes
- 2 Animer ses `View` avec `ObjectAnimator`
- 3 Comment créer ses propres `View`, avec `onDraw()` et `onTouchEvent()`

ConstraintLayout

- 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



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);
```


Exemples de contraintes

- Aligner le bord gauche de view1 et view2 :

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

Exemples de contraintes

- Aligner le bord gauche de view1 et view2 :

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

- Centrer verticalement view :

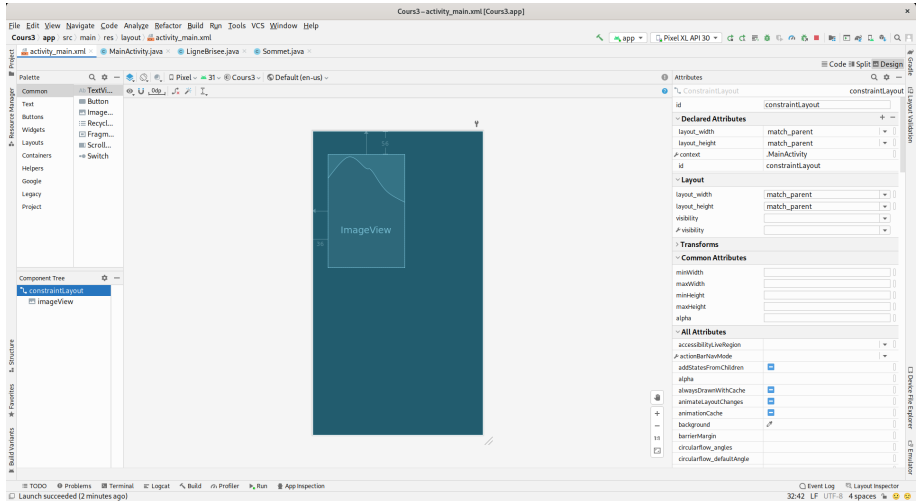
```
constraintSet.connect(  
    view.getId(), ConstraintSet.TOP,  
    constraintLayout.getId(), ConstraintSet.TOP,  
    0);  
constraintSet.connect(  
    view.getId(), ConstraintSet.BOTTOM,  
    constraintLayout.getId(), ConstraintSet.BOTTOM,  
    0);
```

Exemples de contraintes (suite)

- 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



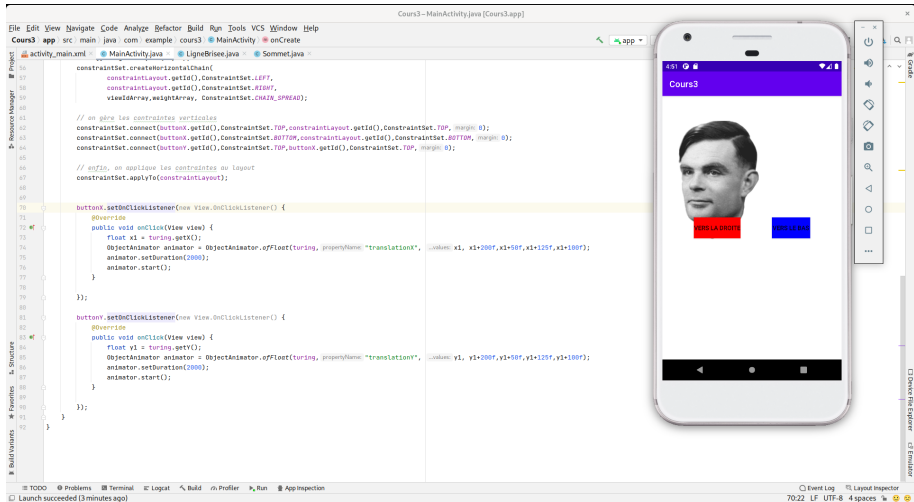
Ajout de Views et de leurs contraintes

The screenshot displays the Android Studio IDE with the `MainActivity.java` file open. The code implements the `onCreate` method, which sets up the UI for an app named "Cours3". It includes a `TextView` with a portrait of Albert Einstein, two `Button` objects labeled "Vers la droite" and "Vers la gauche", and a `TextView` with the text "Ligne Brisee". The layout is defined using `ConstraintLayout` with various constraints and weights to position the views.

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    turing = findViewById(R.id.imageView);  
    constraintLayout = findViewById(R.id.constraintLayout);  
  
    // on crée et instancie nos deux boutons  
    buttonX = new Button(context, this); buttonY = new Button(context, this);  
    buttonX.setText("Vers la droite"); buttonY.setText("Vers la gauche");  
    buttonX.setBackgroundColor(Color.RED); buttonY.setBackgroundColor(Color.BLUE);  
    // on crée notre ligne brisée  
    ligneBrisée = new TextView(context, this, Color.BLACK);  
  
    // on ajoute les views au layout  
    constraintLayout.addView(ligneBrisée);  
    constraintLayout.addView(buttonX);  
    constraintLayout.addView(buttonY);  
  
    // on donne des identifiants à nos views  
    ligneBrisée.setId(View.generateViewId());  
    buttonX.setId(View.generateViewId());  
    buttonY.setId(View.generateViewId());  
  
    // on crée un ConstraintSet  
    ConstraintSet constraintSet = new ConstraintSet();  
    constraintSet.clone(constraintLayout);  
  
    // on crée une chaîne horizontale entre les deux boutons  
    int[] viewIdArray = {buttonX.getId(), buttonY.getId()};  
    float[] weightArray = {1f, 1f};  
    constraintSet.createHorizontalChain(  
        constraintLayout.getId(), ConstraintSet.LEFT,  
        constraintLayout.getId(), ConstraintSet.RIGHT,  
        viewIdArray, weightArray, ConstraintSet.CHAIN_SPREAD);  
  
    // on gère les contraintes verticales  
    constraintSet.connect(buttonX.getId(), ConstraintSet.TOP, constraintLayout.getId(), ConstraintSet.TOP, margin: 0);  
    constraintSet.connect(buttonY.getId(), ConstraintSet.TOP, constraintLayout.getId(), ConstraintSet.TOP, margin: 0);  
    constraintSet.connect(buttonX.getId(), ConstraintSet.BOTTOM, constraintLayout.getId(), ConstraintSet.BOTTOM, margin: 0);  
    constraintSet.connect(buttonY.getId(), ConstraintSet.BOTTOM, constraintLayout.getId(), ConstraintSet.BOTTOM, margin: 0);  
  
    // enfin, on applique les contraintes au layout  
    constraintSet.applyTo(constraintLayout);  
}
```

The right side of the interface shows a preview of the app's UI. It features a purple header with the text "Cours3", a portrait of Albert Einstein, and two buttons labeled "Vers la droite" (red) and "Vers la gauche" (blue). The status bar at the top of the preview shows the time as 6:45 and battery level at 100%.

Turing prend le large



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

Project

Resource Manager

```
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
```

LigneBrisee

Cours3 -- LigneBrisee.java [Cours3.app]

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

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

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

Resource Manager

```
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
35
36
37
38
39
40
41
42
43
44
45     // callback appelé à chaque mouvement de l'utilisateur
46     @Override
47     public boolean onTouchEvent(MotionEvent event) {...}
48
49 }
```

Structure

Favorites

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

- Top Bar:** File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help.
- Project View:** Shows the project structure for 'Cours3 app' with files like 'src/main/java/com/example/cours3/LigneBrisee.java' and 'Sommet.java'.
- Code Editor:** Displays the Java code for 'LigneBrisee.java'. The code includes a canvas drawing method and a touch event listener.
- Preview:** A smartphone mockup showing the app's UI. The app has a purple header with the text 'Cours3'. The main content area shows a drawing of a face with a network of points and lines overlaid. Two red and blue labels are visible: 'VERS LA DROITE' and 'VERS LA GAUCHE'.
- Bottom Bar:** Contains tabs for TOOLS, PROBLEMS, TERMINAL, LOGCAT, BUILD, PROFILER, RUN, and APP INSPECTION. The status bar at the bottom indicates 'Launch succeeded (a minute ago)' and the time '34:29'.

```
// onDraw est appelée à chaque fois que la vue est dessinée
@Override
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 source = sommetsSpeciaux.get(i);
        canvas.drawCircle(source.getX(), source.getY(), radius, pf, paint);
    }
}

// callback appelé à chaque mouvement de l'utilisateur
@Override
public boolean onTouchEvent(MotionEvent event) {
    super.onTouchEvent(event);
    Sommet sommet;
    switch(event.getAction()) {
        // cas où il y a une discontinuité
        case MotionEvent.ACTION_DOWN:
        case MotionEvent.ACTION_UP:
            sommet = new Sommet(event.getX(), event.getY());
            sommets.add(sommet);
            sommetsSpeciaux.add(sommet);
            break;
        // cas où un mouvement est détecté
        // on récupère alors le nombre de points intermédiaires via getHistorySize()
        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:
            // on retire la View -> appel à onDraw()
            invalidate();
            return true;
    }
}
```

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>setStrokeWidth()</code>	<code>drawLine()</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>setStrokeWidth()</code>	<code>drawLine()</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;
}
```

