

Programmation mobile

Cours 3 : [View](#)

Julien Grange <julien.grange@acl.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...

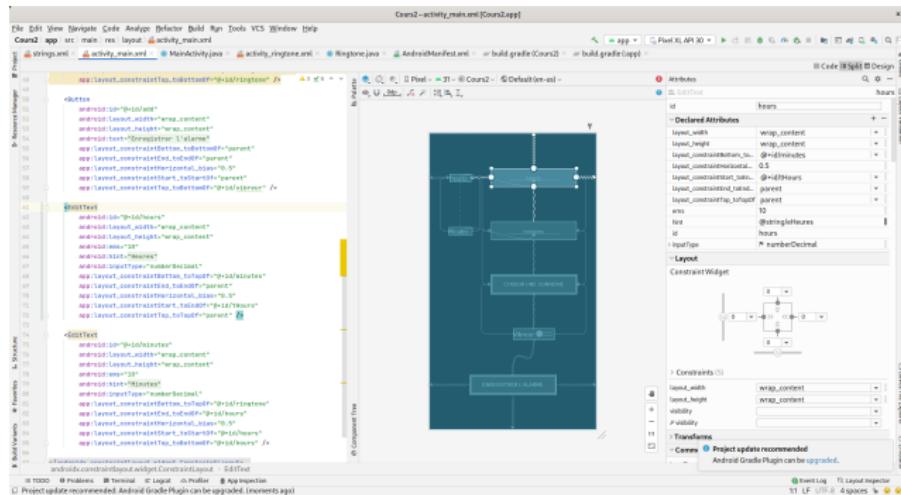
- ① `ConstraintLayout` : comment ajouter dynamiquement des `View` et en ajuster les contraintes
- ② Animer ses `View` avec `ObjectAnimator`
- ③ Comment créer ses propres `View`, avec `onDraw()` et `onTouchEvent()`

ConstraintLayout

- Évite les imbrications de `ViewGroup` → réduit la profondeur de l'arbre.
- Ses enfants sont liés via des *constraints* :
 - 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 :

- ➊ Créer un **ConstraintSet** :

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

- ➋ Ajouter des contraintes :

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

- ➌ 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);
```

- Centerer verticalement view :

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

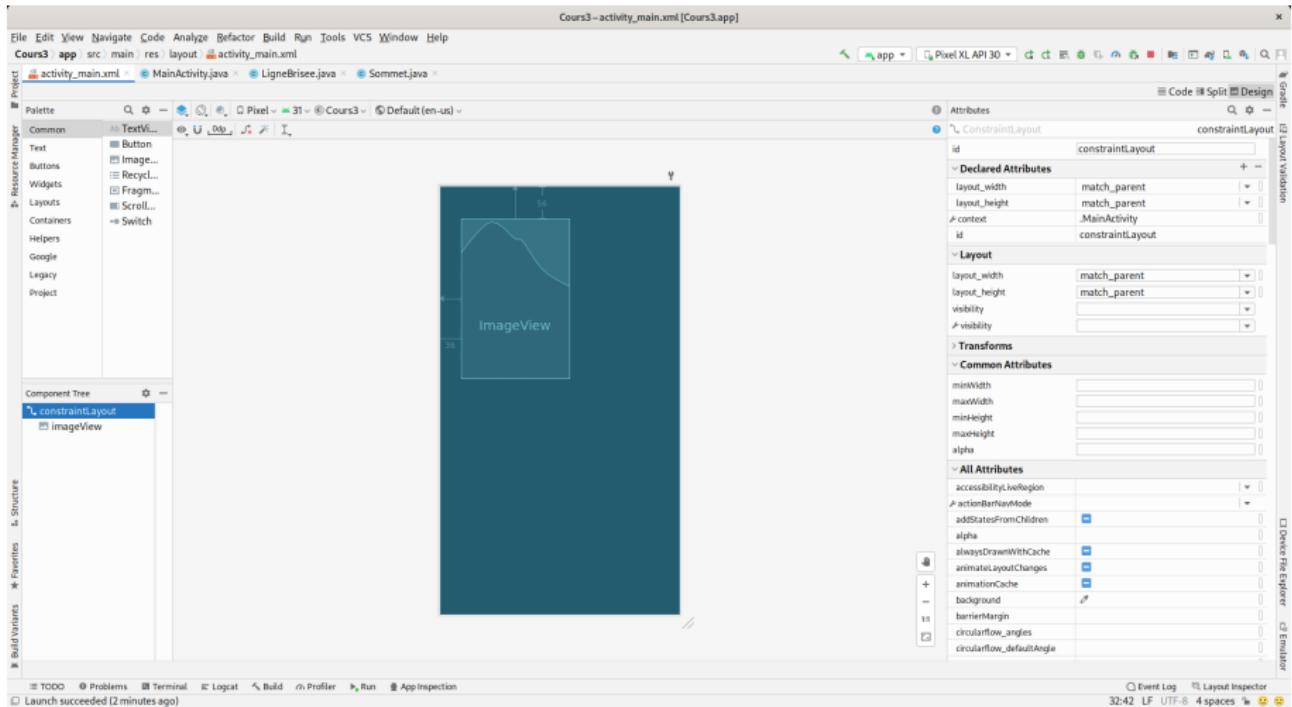
Exemples de contraintes (suite)

- 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

Cours3 - MainActivity.java [Cours3.app]

```
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 instance nos deux boutons
    buttonX = new Button(this); buttonY = new Button(this);
    buttonX.setText("Vers la droite"); buttonY.setText("Vers la bas");
    buttonX.setBackgroundColor(Color.RED); buttonY.setBackgroundColor(Color.BLUE);
    // on crée notre ligne brisée
    ligneBrisee = new LigneBrisee(this,Color.BLACK);

    // on ajoute les views au layout
    constraintLayout.addView(ligneBrisee);
    constraintLayout.addView(buttonX);
    constraintLayout.addView(buttonY);

    // on donne des identifiants à nos views
    ligneBrisee.setId(View.generateViewId());
    buttonX.setId(View.generateViewId());
    buttonY.setId(View.generateViewId());

    // on crée un ConstraintSet
    ConstraintSet constraintSet = new ConstraintSet();
    constraintSet.clone(constraintLayout);

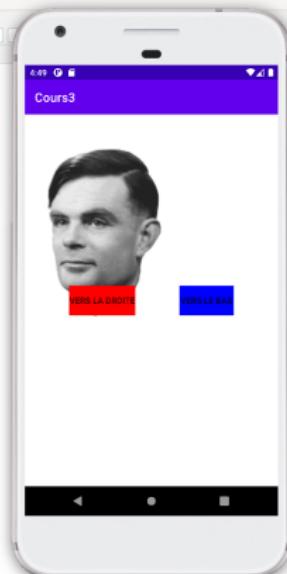
    // on crée une chaîne horizontale entre les deux boutons
    int[] viewIdArray = {buttonX.getId(),buttonY.getId()};
    float[] weightArray = {0f,0f};
    constraintSet.createHorizontalChain(
        constraintLayout.getId(),ConstraintSet.LEFT,
        constraintLayout.getId(),ConstraintSet.RIGHT,
        viewIdArray,weightArray, ConstraintSet.CHAIN_SPREAD);

    // on gère les contraintes verticales
    constraintSet.connect(buttonX.getId(),constraintSet.TOP,constraintLayout.getId(),ConstraintSet.TOP, margin: 0);
    constraintSet.connect(buttonX.getId(),ConstraintSet.BOTTOM,constraintLayout.getId(),ConstraintSet.BOTTOM, margin: 0);
    constraintSet.connect(buttonY.getId(),ConstraintSet.TOP,buttonX.getId(),ConstraintSet.TOP, margin: 0);

    // enfin, on applique les contraintes au layout
    constraintSet.applyTo(constraintLayout);
```

Julien Grange

Programmation mobile



Event Log Layout Inspector

70:22 LF UTF-8 4 spaces

Turing prend le large

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

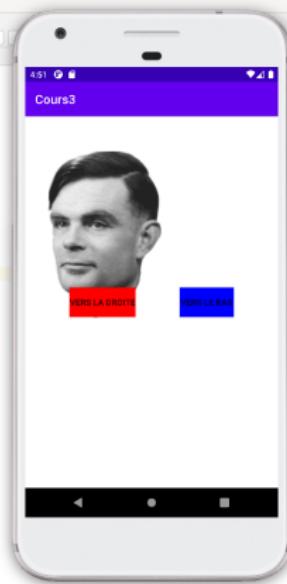
Cours3 app src main java com.example.cours3 MainActivity onCreate

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

```
constraintSet.createHorizontalChain(  
        constraintLayout.getGid(), ConstraintSet.LEFT,  
        constraintLayout.getGid(), 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(buttonX.getId(), ConstraintSet.BOTTOM, constraintLayout.getId(), ConstraintSet.BOTTOM, margin: 0);  
constraintSet.connect(buttonY.getId(), ConstraintSet.TOP, buttonX.getId(), ConstraintSet.TOP, margin: 0);  
  
// enfin, on applique les contraintes au layout  
constraintSet.applyTo(constraintLayout);  
  
buttonX.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        float x1 = turing.getX();  
        ObjectAnimator animator = ObjectAnimator.ofFloat(turing, propertyName: "translationX",  
            animator.setDuration(2000);  
        animator.start();  
    }  
});  
  
buttonY.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        float y1 = turing.getY();  
        ObjectAnimator animator = ObjectAnimator.ofFloat(turing, propertyName: "translationY",  
            animator.setDuration(2000);  
        animator.start();  
    }  
});  
};
```

4:51 9:11 4G

Cours3



Julien Grange

Programmation mobile

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();
```

Suivi du toucher

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

Suivi du toucher

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

The screenshot shows the Android Studio interface with the following details:

- Menu Bar:** File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help.
- Project Bar:** Cours3 > app > src > main > java > com > example > cours3 > Sommet
- Code Editor:** The file `Sommet.java` is open. The code defines a class `Sommet` with private final attributes `x` and `y`, a constructor that initializes these attributes, and methods to get the values of `x` and `y`. The code is as follows:

```
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
```

The code editor has tabs for `activity_main.xml`, `MainActivity.java`, `LigneBrisee.java`, and `Sommet.java`, with `Sommet.java` currently selected. The left sidebar shows the **Project** and **Resource Manager** sections.

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 < MainActivity.java < LigneBrisee.java < Sommet.java <
1 package com.example.cours3;
2
3 import ...
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     // callback appelé à chaque mouvement de l'utilisateur
35     @Override
36     public boolean onTouchEvent(MotionEvent event) {...}
37
38 }
```

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

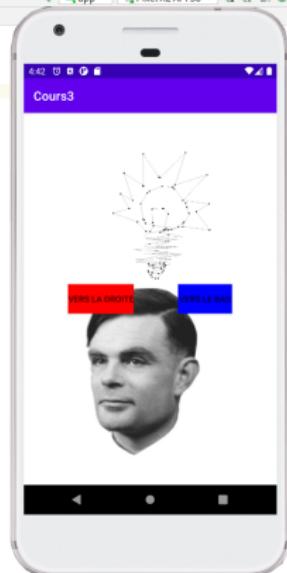
30 // onDraw est appelée à chaque fois que la vue est dessinée
31 @Override
32 protected void onDraw(Canvas canvas) {
33     super.onDraw(canvas);
34     for (int i = 0; i < sommets.size()-1; i++) {
35         Sommet source = sommets.get(i);
36         Sommet cible = sommets.get(i+1);
37         canvas.drawLine(source.getTx(), source.getTy(), cible.getTx(), cible.getTy(), paint);
38     }
39     for (int i = 0; i < sommetsSpeciaux.size()-1; i++) {
40         Sommet source = sommetsSpeciaux.get(i);
41         canvas.drawCircle(source.getTx(), source.getTy(), radius, 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.getTx(), event.getTy());
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     }
68     // on retrace la View -> appel à onDraw()
69     invalidate();
70 }
71
72 // Launch succeeded (a minute ago)
```

42% 0% 27% 34% 41% 48% 55% 62% 69% 76% 83% 90% 97% 100%

Pixel XL API 30

Cours3

VERS LA DROITE VERS LA GAUCHE



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**)

Paint	Canvas
setColor()	drawLine()
setStrokeWidth()	drawCircle()

```
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;
}
```

