Programmation mobile

Cours 3 : `View`

Julien Grange <julien.grange@lacl.fr>

Mardi 8 octobre 2024

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL

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...

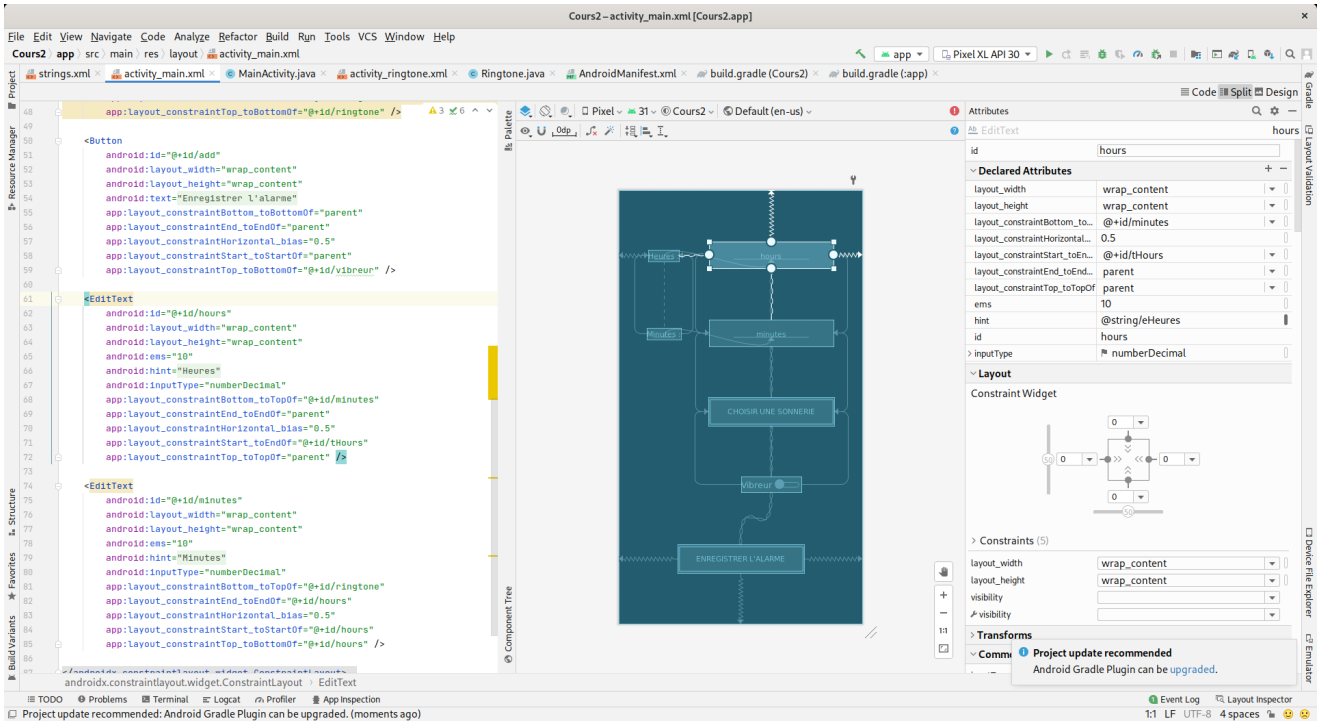
1. `ConstraintLayout` : comment ajouter dynamiquement des `View` et en ajuster les contraintes
2. Animer ses `View` avec `ObjectAnimator`
3. Comment créer ses propres `View`, avec `onDraw()` et `onTouchEvent()`

`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` :

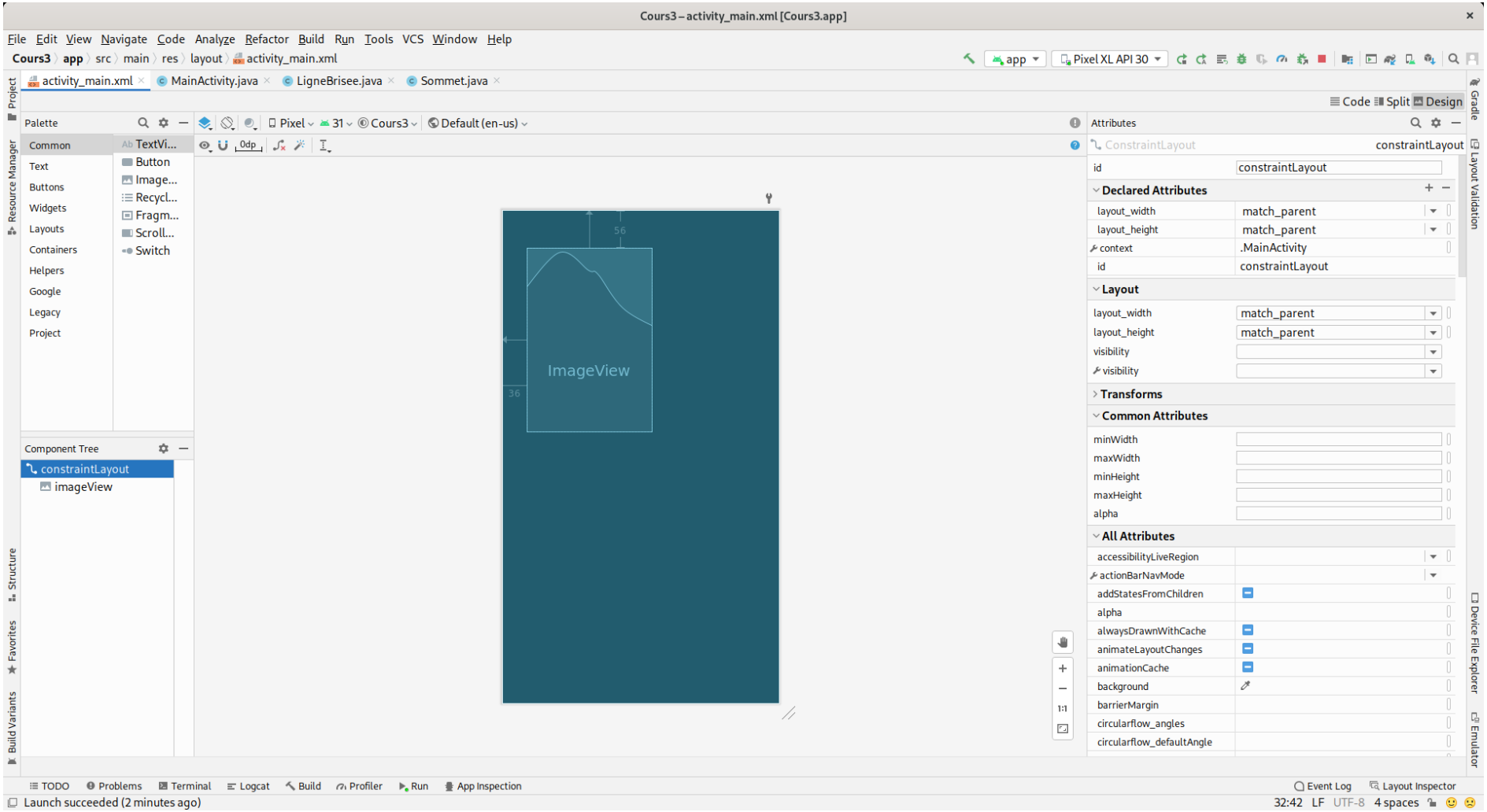
```
constraintSet.connect(
    view.getId(),ConstraintSet.TOP,
    constraintLayout.getId(),ConstraintSet.TOP,
    0);
constraintSet.connect(
    view.getId(),ConstraintSet.BOTTOM,
    constraintLayout.getId(),ConstraintSet.BOTTOM,
    0);
```

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

Cours3 – MainActivity.java [Cours3.app]

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

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

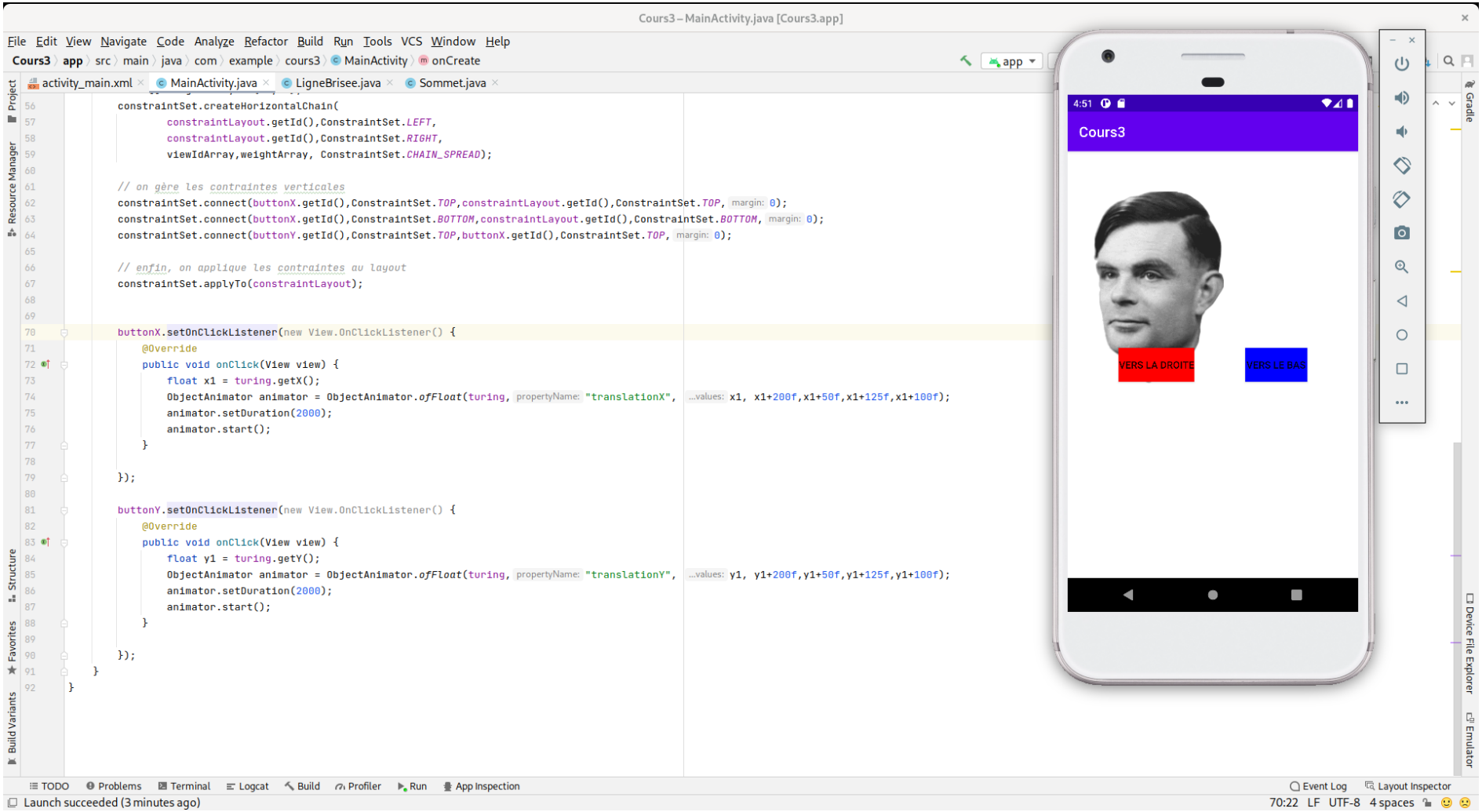
activity_main.xml | MainActivity.java | LigneBrisee.java | Sommet.java

```java
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 le bas");
    buttonX.setBackgroundColor(Color.RED); buttonY.setBackgroundColor(Color.BLUE);
    // on crée notre ligne brisée
    ligneBrisee = new LigneBrisee( context: 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);
```

TODO | Problems | Terminal | Logcat | Build | Profiler | Run | App Inspection | Event Log | Layout Inspector

Launch succeeded (moments ago) 70:22 LF UTF-8 4 spaces

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();
```

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

Sommet

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 | MainActivity.java | LigneBrisee.java | Sommet.java

Project | Resource Manager

```java
package com.example.cours3;

public class Sommet {
    private final float x,y;

    public Sommet(float x, float y){
        this.x = x;
        this.y = y;
    }

    public float getX(){ return x;}
    public float getY(){ return y;}

}
```

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

app

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

```java
package com.example.cours3;

import ...

public class LigneBrisee extends View {

    private final static int default_color = Color.BLACK;
    private Paint paint = new Paint();
    // liste des sommets
    private ArrayList<Sommet> sommets = new ArrayList<Sommet>();
    // listes des sommets où le doigt a été levé
    private ArrayList<Sommet> sommetsSpeciaux = new ArrayList<Sommet>();

    public LigneBrisee(Context context, int color) {
        super(context);
        paint.setColor(color);
    }

    public LigneBrisee(Context context) {
        this(context,default_color);
    }

    // onDraw est appelée à chaque fois que la vue est dessinée
    @Override
    protected void onDraw(Canvas canvas) {...}

    // callback appelé à chaque mouvement de l'utilisateur
    @Override
    public boolean onTouchEvent(MotionEvent event) {...}
}
```

LigneBrisee

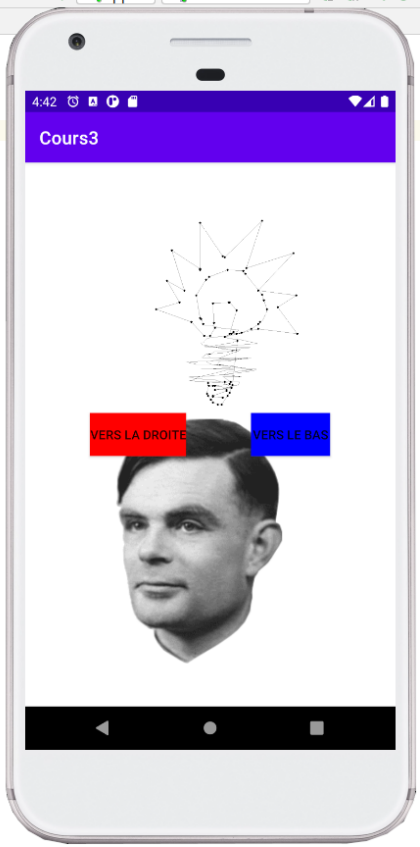
Cours3 – LigneBrisee.java [Cours3.app]

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

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

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

```java
// 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: 4f,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 retrace la View -> appel à onDraw()
    invalidate();
    return true;
}
```



TODO Problems Terminal Logcat Build Profiler Run App Inspection Event Log Layout Inspector

Launch succeeded (a minute ago) 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`)

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

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;
}
```

LigneBrisee