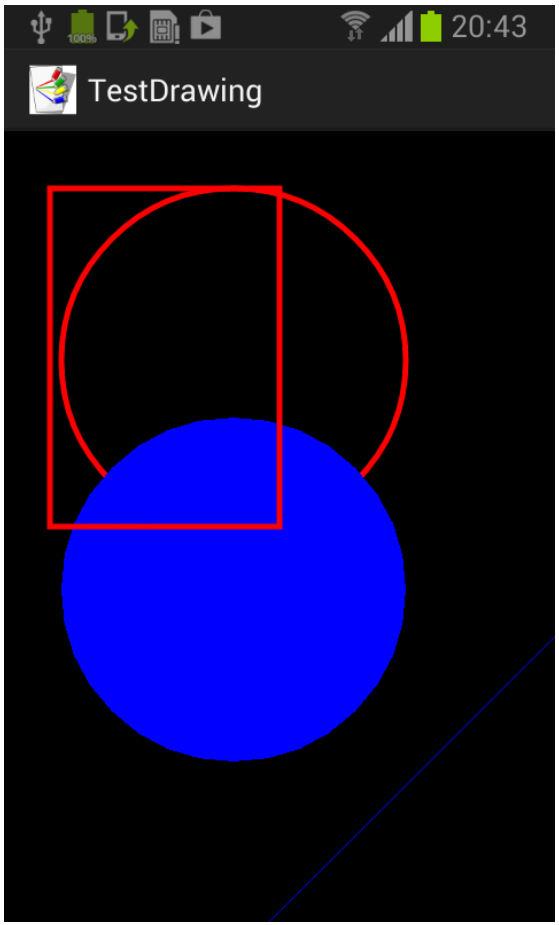
Dessiner et vibrer avec Android



Le dessin

Scénario classique

- fichiers communs : activity_layout.xml, strings.xml et MyActivity.java
- activity_layout.xml : description des éléments qui compose le layout
- MyActivity.java : appel à la méthode setContentView(R.layout.activity_layout)

  $\Rightarrow$ construction de l'interface graphique composée des différents éléments (View) qui la composent et qui sont décrits dans le fichier xml du layout
- qu'en est-il du dessin ?

Le dessin

application de dessin

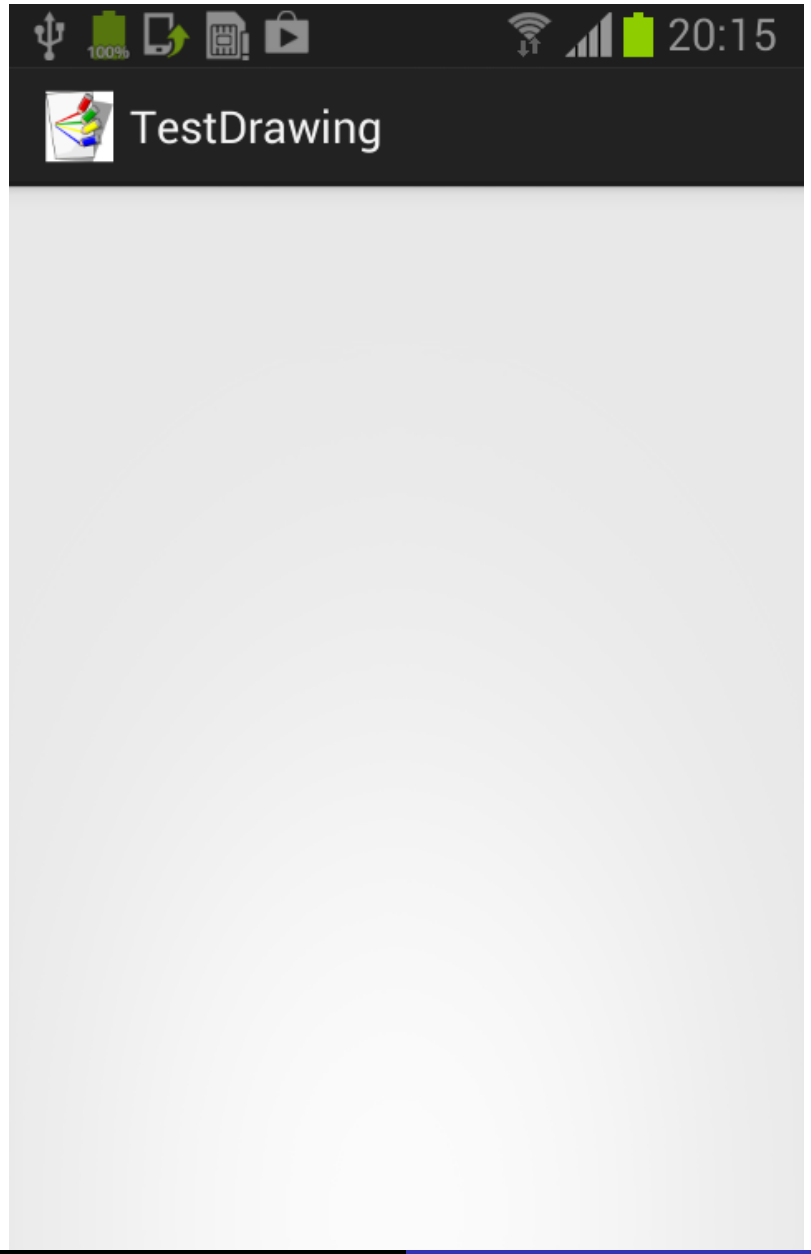
- **VOUS** devez construire la View
- une manière simple de construire une View consiste à étendre la classe View
- instanciez cette classe
- dans le corps de l'Activity effectuez un appel à la méthode setContentView(your_own_view)

Le dessin

code minimaliste pour dessiner

```
public class TestDrawing extends Activity {

    MyOwnView whatIdraw;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        whatIdraw = new MyOwnView(this);
        setContentView(whatIdraw);
    }
}

class MyOwnView extends View {

    public MyOwnView(Context context) {
        super(context);
        setFocusable(true);
    }

    public void onDraw(Canvas canvas) { }

}
```

code minimaliste pour dessiner



Le dessin

trois classes principales

- Bitmap : la surface sur laquelle vous dessinez

  pour créer un bitmap vous pouvez appeler la méthode statique Bitmap.createBitmap() mais la méthode *onDraw()* fournit une Bitmap.
- Canvas : l'objet en charge du dessin

  rectangle, circle, line, text, etc.
- Paint : la manière dont le dessin sera fait ($\sim$ style)

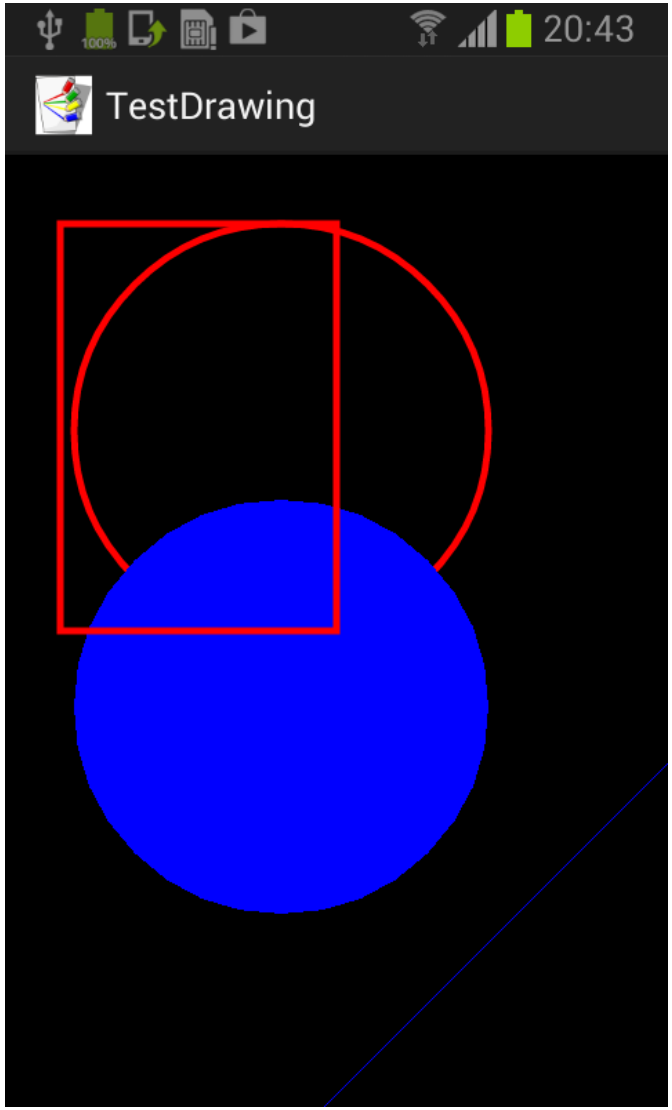
  fill, stroke, colors, etc.

Dessiner avec Style

```
class MyOwnView extends View {

    Paint firstWayToDraw = new Paint();
    Paint secondWayToDraw = new Paint();

    public MyOwnView(Context context) {
        super(context);
        setFocusable(true);
        firstWayToDraw.setColor(Color.RED);
        firstWayToDraw.setAntiAlias(true);
        firstWayToDraw.setStyle(Paint.Style.STROKE);
        firstWayToDraw.setStrokeWidth(5);
        secondWayToDraw.setColor(Color.BLUE);
        secondWayToDraw.setAntiAlias(false);
        secondWayToDraw.setStyle(Paint.Style.FILL_AND_STROKE);
    }
```

Dessiner avec Style

```
public void onDraw(Canvas canvas) {
    canvas.drawColor(Color.BLACK);
    canvas.drawCircle(200, 200, 150, firstWayToDraw);
    canvas.drawCircle(200, 400, 150, secondWayToDraw);
    canvas.drawRect(40, 50, canvas.getWidth()/2, canvas.getHeight()/2, firstWayToDraw)
    canvas.drawLine(900, 20, 20, 900, secondWayToDraw);
}
```


20:43
TestDrawing

Gestion des événements

Event onTouch

- la surface sur laquelle vous dessinez est *"touchable"* et *"clickable"*
- poser un doigt sur l'écran $\rightarrow$ génération d'un MotionEvent
- à un MotionEvent est associé un code d'action :
  - ACTION_DOWN (dès que le doigt touche l'écran),
  - ACTION_UP lorsque le doigt cesse de toucher l'écran,
  - ACTION_MOVE lorsque le doigt se déplace sur l'écran (pour chaque action on peut récupérer les coordonnées x et y)
  - ACTION_CANCEL.

liste des actions

https://developer.android.com/reference/android/view/MotionEvent.html

(constantes)

Gestion des événements

Event onTouch

- nous implémentons les deux interfaces OnTouchListener et OnClickListener
- nous nous restreignons à seulement un objet Paint et à un seul cercle (rayon 50).
- lorsqu'un *Touch Event* est détecté, nous changeons la position du cercle pour le faire coincider avec la position du doigt
- lorsqu'un clic est effectué, nous changeons la couleur du cercle

Gestion des événements

```
public class MyOwnView extends View implements OnTouchListener, OnClickListener
```

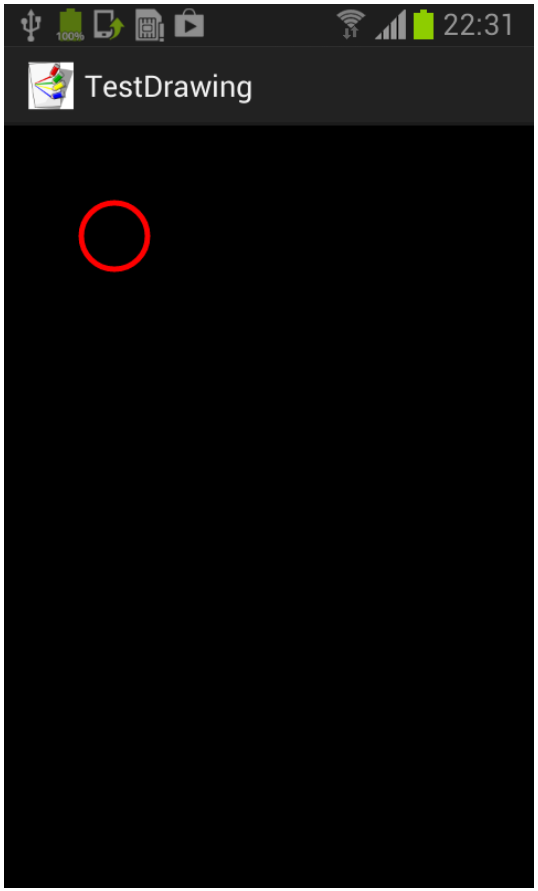
```
    public MyOwnView(Context context) {
        super(context);
        firstWayToDraw.setColor(Color.RED);
        firstWayToDraw.setAntiAlias(true);
        firstWayToDraw.setStyle(Paint.Style.STROKE);
        firstWayToDraw.setStrokeWidth(5);
        this.setOnTouchListener(this);
        this.setOnClickListener(this);
    }
```

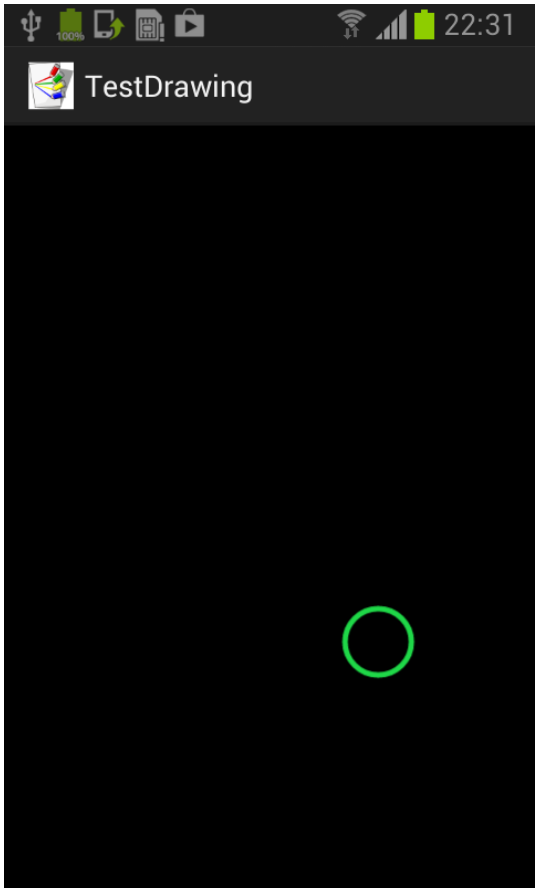
Gestion des événements

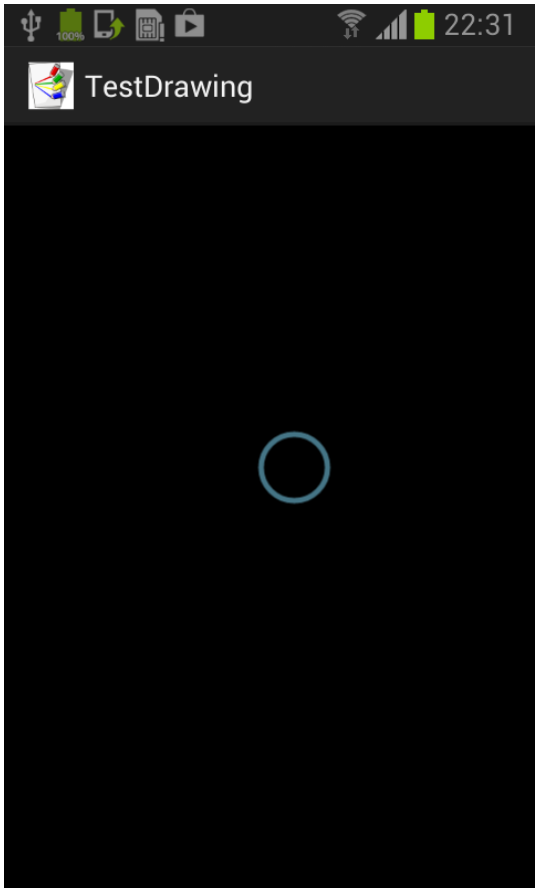
```
        public void onDraw(Canvas canvas) {
            canvas.drawColor(Color.BLACK);
            canvas.drawCircle(x, y, radius, firstWayToDraw);
        }


public boolean onTouch(View v, MotionEvent event) {
    x = event.getX();
    y = event.getY();
    this.invalidate();
    return false;
}

public void onClick(View v) {
    firstWayToDraw.setColor(Color.rgb(rd.nextInt(200), rd.nextInt(240), rd.nextInt(180
    this.invalidate();
}
```

- this.invalidate() ; indique à l'objet qu'il doit être redessiné
- return false ; à la fin du code de onTouch() indique que l'événement n'a pas été consommé, il sera donc retransmis aux autres méthodes (onClick() en particulier))

Gestion des événements



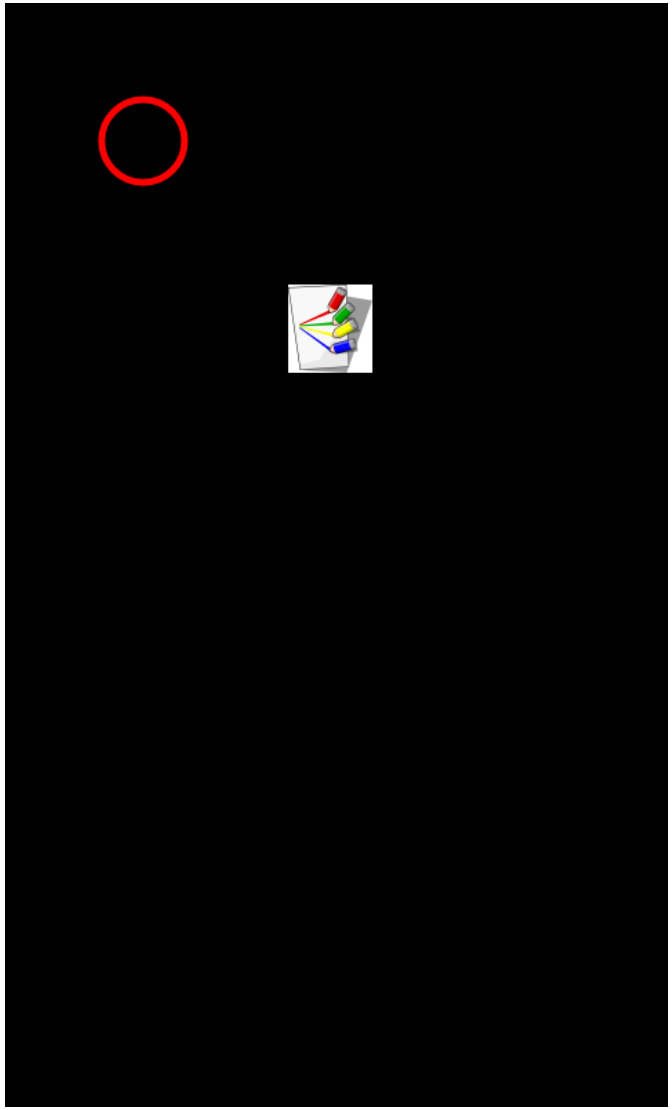




Ajouter une image

```
public void onDraw(Canvas canvas) {
    canvas.drawColor(Color.BLACK);
    Bitmap bmp = BitmapFactory.decodeResource(getResources(),R.drawable.ic_launcher);
    canvas.drawBitmap(bmp,x+100,y+100,null);
    canvas.drawCircle(x, y, radius, firstWayToDraw);
}
```

Ajouter une image



Gestion du vibreur

Vibrator

- la plupart des matériels sont équipés avec un mécanisme qui génère des vibrations
- cette fonctionnalité est accessible par un appel à la méthode getSystemService(Context.VIBRATOR_SERVICE)
- l'appel à getSystemService() retourne une référence non nulle à une instance de Vibrator qui gère le vibreur
- Si vous souhaitez savoir si votre machine contient un vibreur au niveau du matériel, vous pouvez invoquer la méthode hasVibrator() sur l'instance de Vibrator obtenu par l'appel à getSystemService().

Gestion du vibreur

- l'utilisation du vibreur nécessite une autorisation il est donc nécessaire d'ajouter la ligne correspondante dans le fichier AndroidManifest.xml

  <uses-permission android :name="android.permission.VIBRATE" / >

a minima

```
import android.os.Vibrator;
Vibrator vibor = (Vibrator)getSystemService(Context.VIBRATOR_SERVICE);
```

Gestion du vibreur

Vibrer oui, mais comment ?

- il existe plusieurs façon d'utiliser le vibreur
- vibration simple : `vibor.vibrate(500);` (une demi seconde)
- vibrations multiples selon un motif prédéfini :

  ```
  int pattern[] = {100,200,100,500,100,200};
  vibor.vibrate(pattern,-1);
  ```
  (silence durant 100ms, puis vib. 200ms, silence 100, etc. et pas de répétition (-1))
- attention !! si vous optez pour une répétition du motif il faut penser à l'arrêter
- vibrations multiples avec répétition sans fin :

  ```
  int pattern[] = {100,200,100,500,100,200};
  vibor.vibrate(pattern,0);
  ```
  la répétition se fait pour le motif entier

  `vibor.vibrate(pattern,3);` la répétition se fera à partir de l'entier situé à l'indice 3 du tableau pattern, c'est-à-dire à partir du 500

  `vibor.cancel();` pour arrêter la répétition

Gestion du vibreur

Vibrer un nombre *n* de fois ?

- plus compliqué qu'il n'y paraît de prime abord
- il semblerait (ce n'est pas très bien documenté) que l'appel à la méthode `vibrate()` ne déclenche le vibreur que si celui-ci ne vibre pas déjà

  $\rightarrow$ ainsi si pour faire vibrer votre matériel 5 fois vous utilisez une structure du type ci-dessous, cela ne fonctionnera pas :

```
int[] pattern = {100,200,100,500}
for(int n=0 ;n<5 ;n++) {
    vibreur.vibrate(pattern,-1) ;
}
```

- en effet, après le premier appel à la méthode `vibreur.vibrate()`, les autres appels seront sans effet puisque le vibreur vibrera déjà
- Quelle solution peut-on envisager ?

Gestion du vibreur

Vibrer un nombre *n* de fois ?

- il faut attendre que le vibreur ait terminé de vibrer avant d'effectuer un autre appel à la méthode `vibrate()`

```
int[] pattern = {100,200,100,500}
int delaiVibration = 100+200+100+500;
for(int n=0;n<5;n++) {
    vibreur.vibrate(pattern,-1);
    try {
        Thread.sleep(delaiVibration);
    } catch(InterruptedException ie) {}
}
```

- fonctionnera sans doute, **mais...**

Vibrer un nombre *n* de fois ?

- ..., **mais...** le problème est que cette méthode est exécutée dans le contexte du main UI Thread ce qui n'est pas une bonne pratique, il faut lui préférer une méthode qui alloue un nouveau thread pour effectuer cette tâche particulière :

```
int[] pattern = {100,200,100,500}
int delaiVibration = 100+200+100+500;
new Thread(new Runnable() {
        public void run() {
                for(int n=0;n<5;n++) {
                    vibreur.vibrate(pattern,-1);
                    try {
                        Thread.sleep(delaiVibration);
                    } catch(InterruptedException ie) {}
                }
        }
}).start();
```

Références

Dessin

- https://developer.android.com/reference/android/view/View
- https://developer.android.com/reference/android/graphics/Canvas
- https://developer.android.com/reference/android/graphics/Bitmap
- https://developer.android.com/reference/android/graphics/Paint
- https://developer.android.com/reference/android/graphics/Paint.Style

Evénements

- https://developer.android.com/reference/android/view/View.OnClickListener
- https://developer.android.com/reference/android/view/View.OnTouchListener
- https://developer.android.com/reference/android/view/MotionEvent

Vibreur

- https://developer.android.com/reference/android/os/Vibrator