Views & Graphics

Mike Gorünov, an Android & JVM freelancer, consultant, and mentor

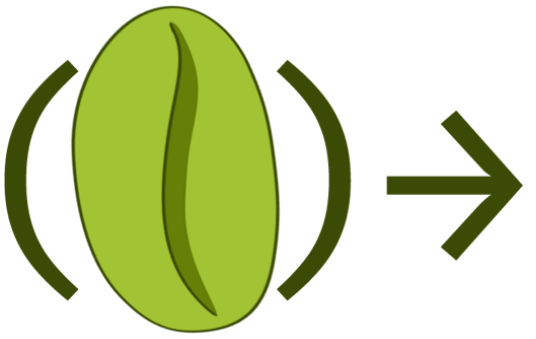
 @Miha-x64

 @miha_x64

 @Harmonizr at android_ru, kotlin_lang, kotlin_lychee, spbpeerlab

@android_broadcast

@androidacademyspb



@javanese_online

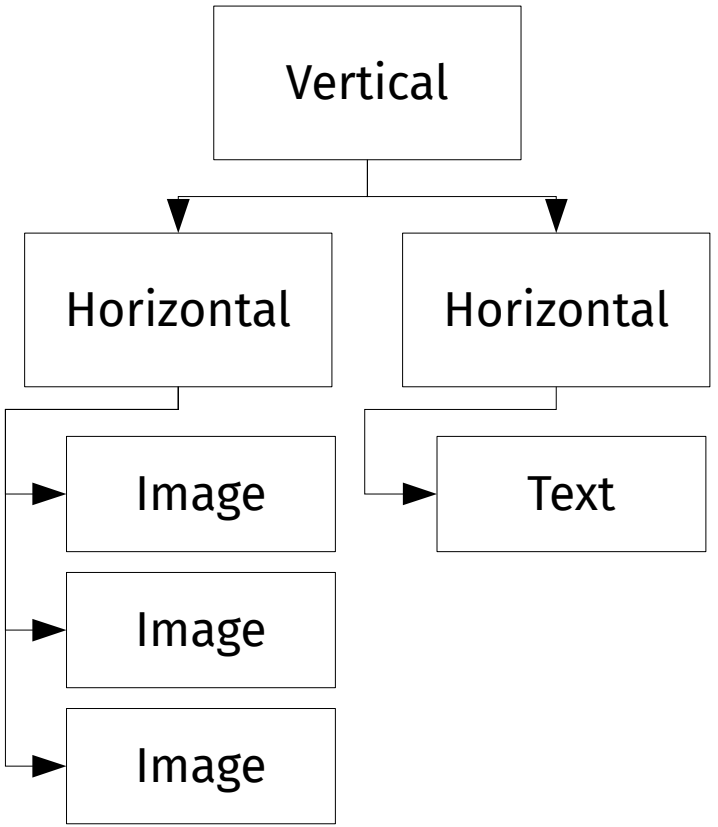
Views & Graphics

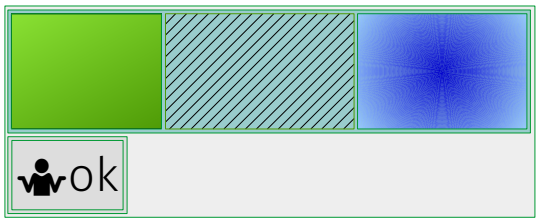
- View passes: measure, layout, draw
- custom measure, custom layout
- drawing: Drawables, Canvas, and Paint
- RecyclerView.ItemDecoration and FontMetrics
- Shaders, Paths, xfermode, SW and HW layers
- animated drawables
- View post-processing

View passes

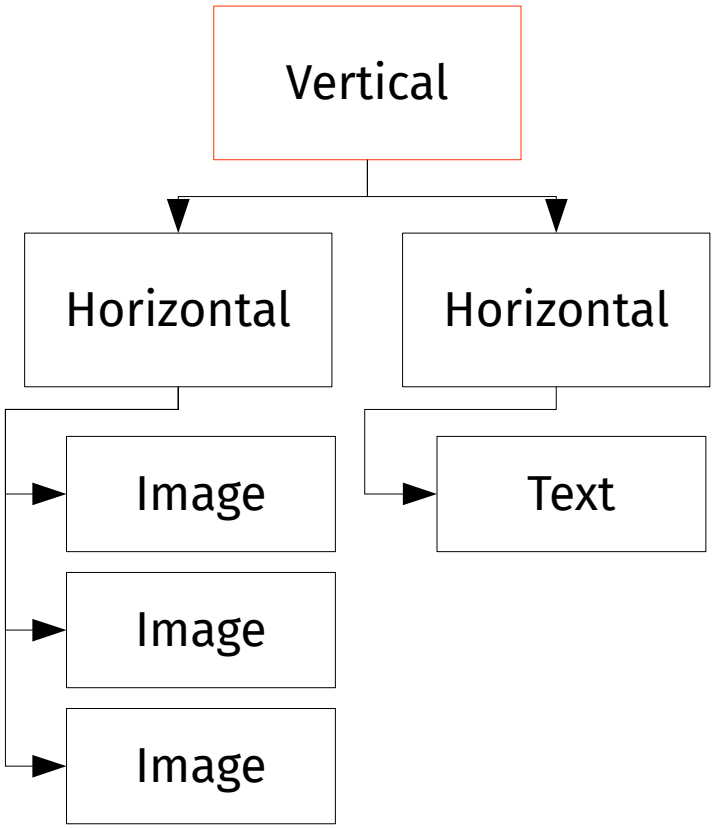
- (on)measure
- (on)layout
- (on)draw

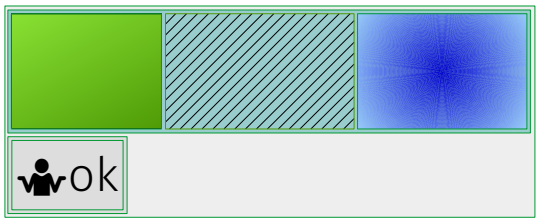
View passes



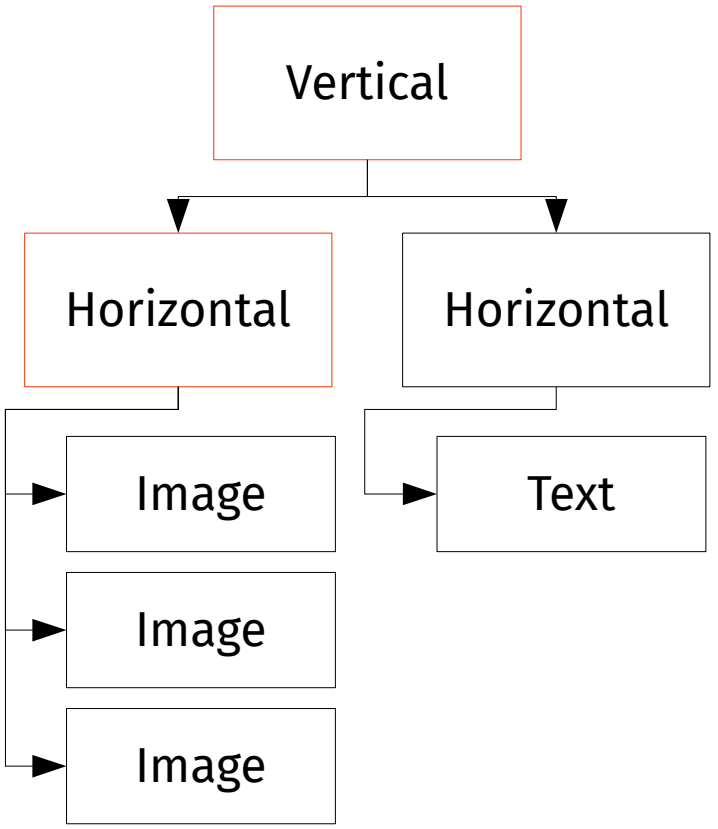


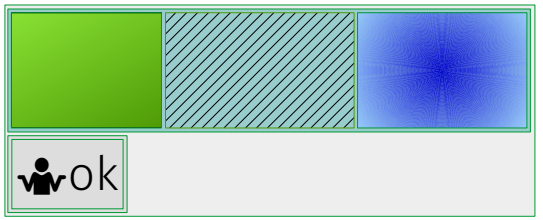
View#measure



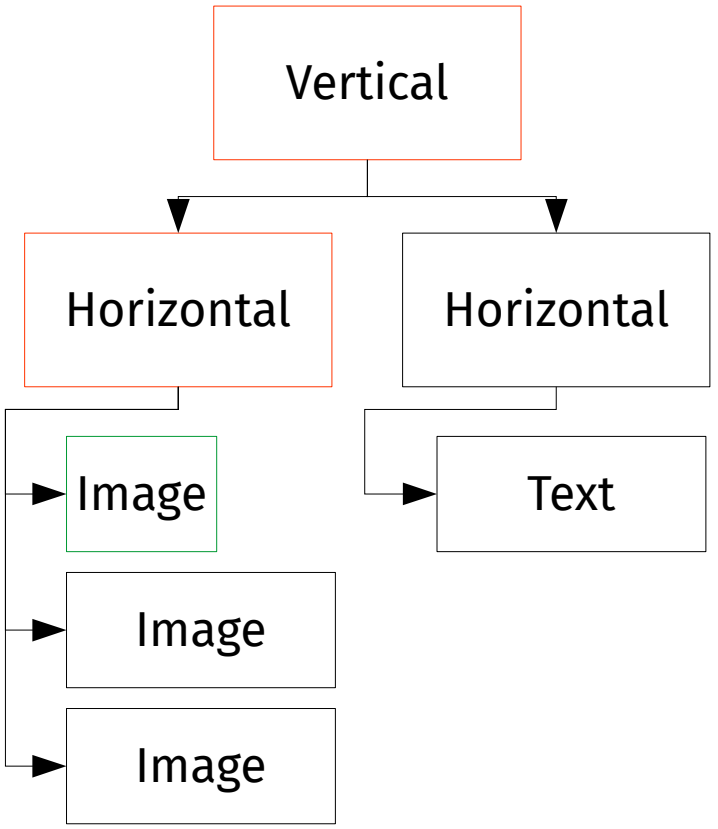


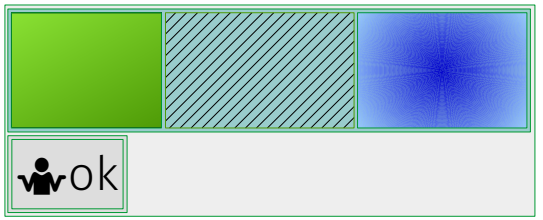
View#measure



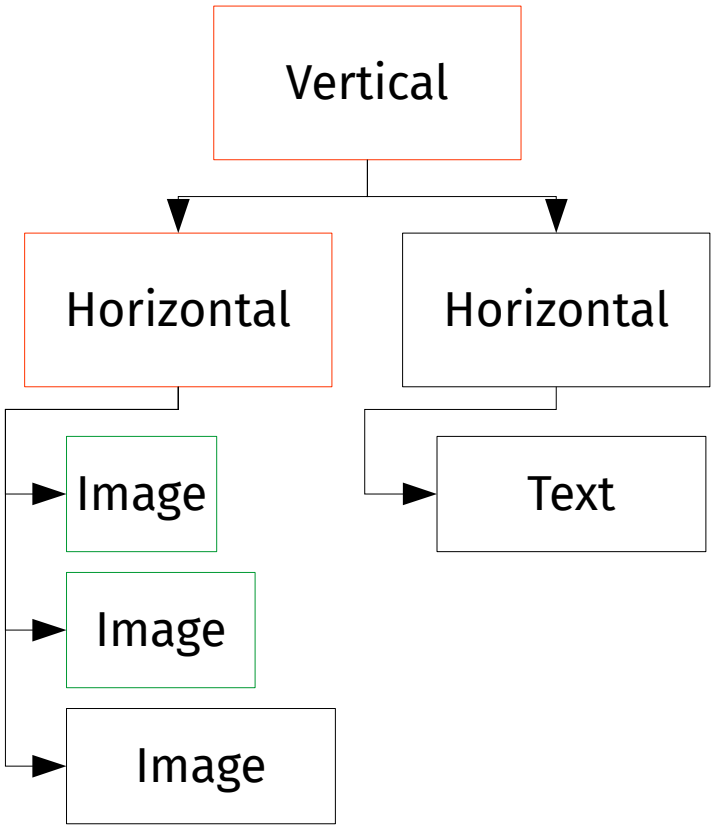


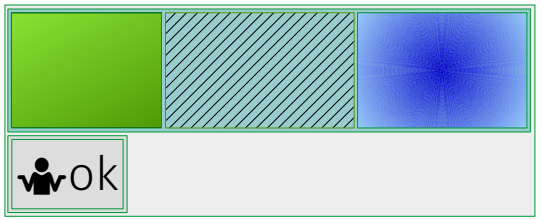
View#measure



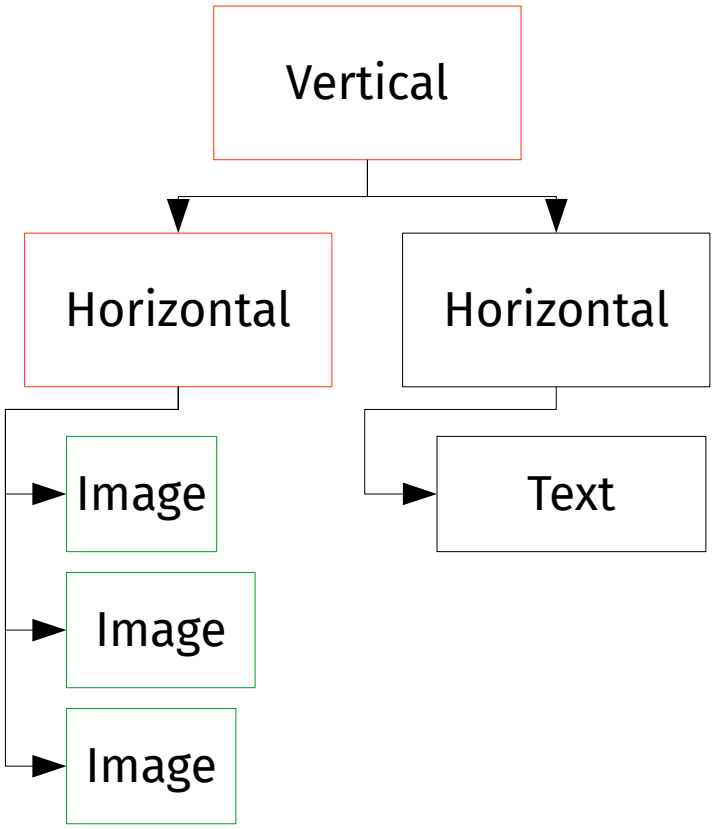


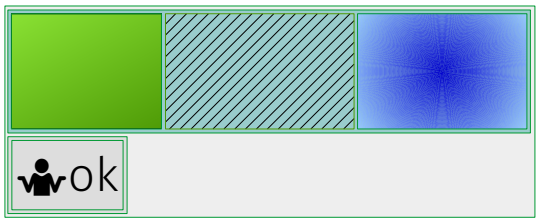
View#measure



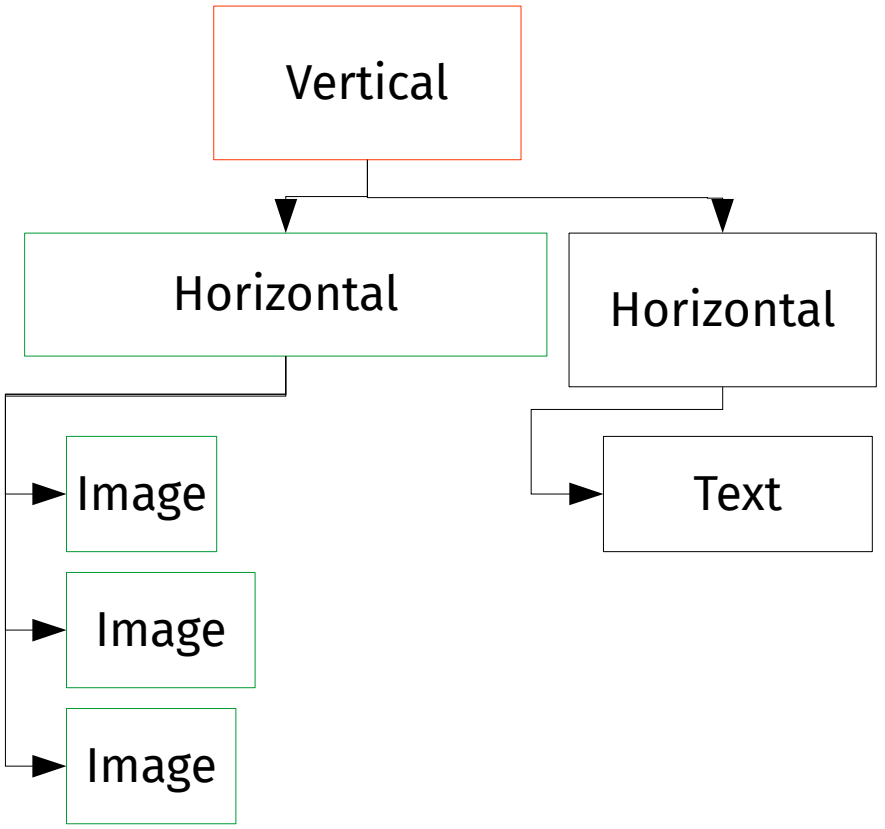


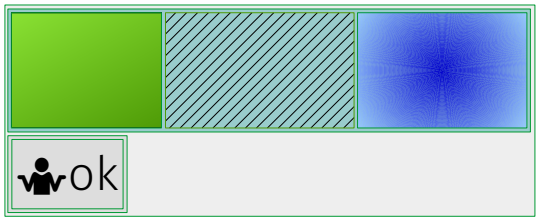
View#measure



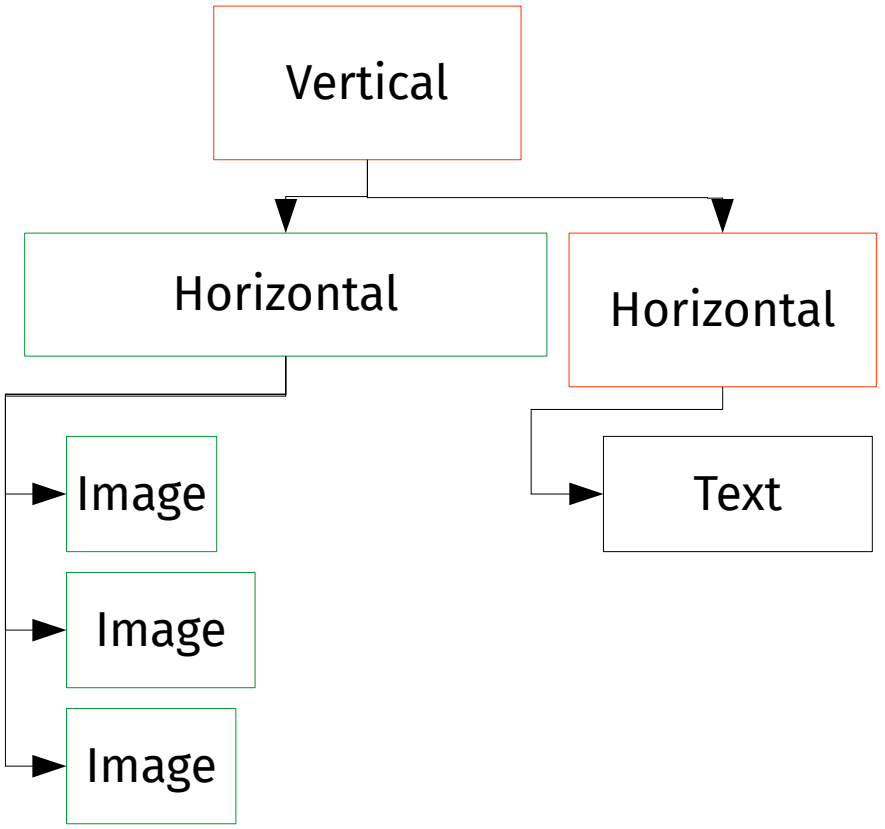


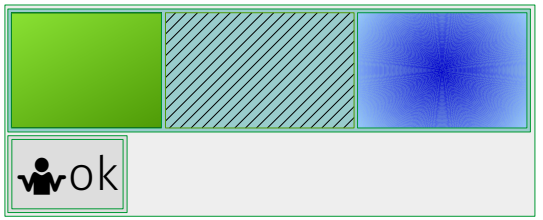
View#measure



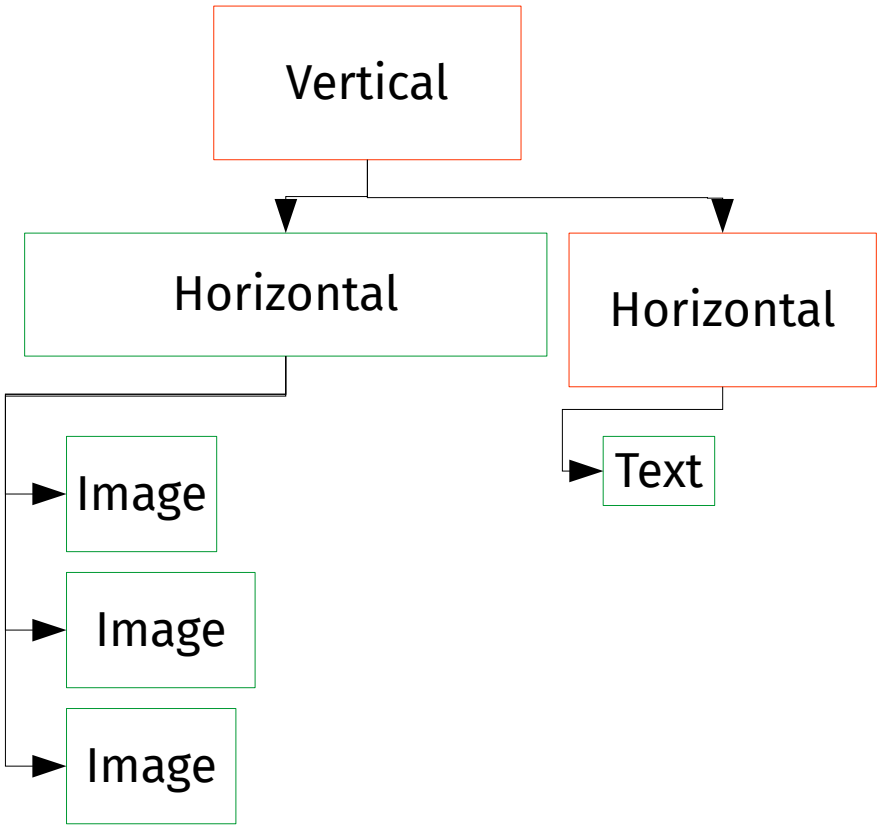


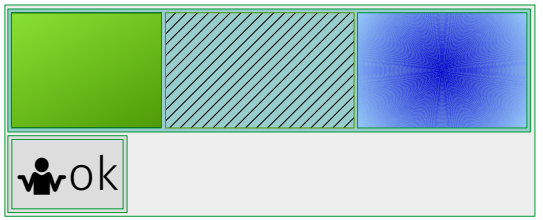
View#measure



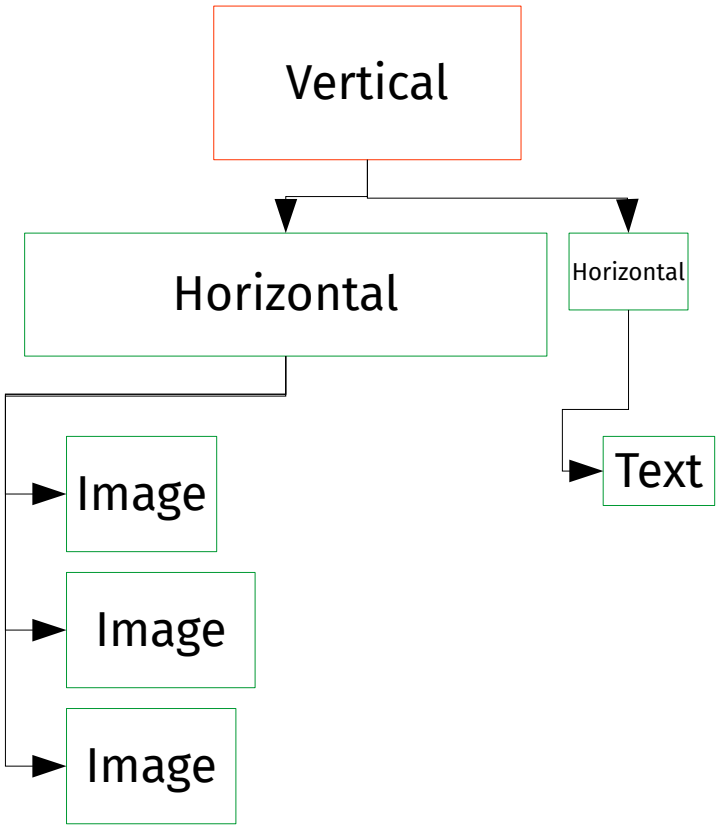


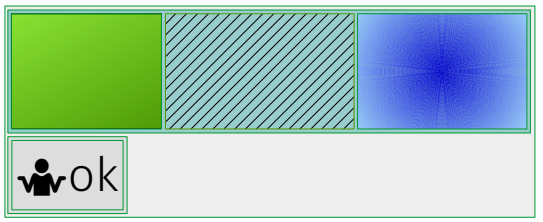
View#measure



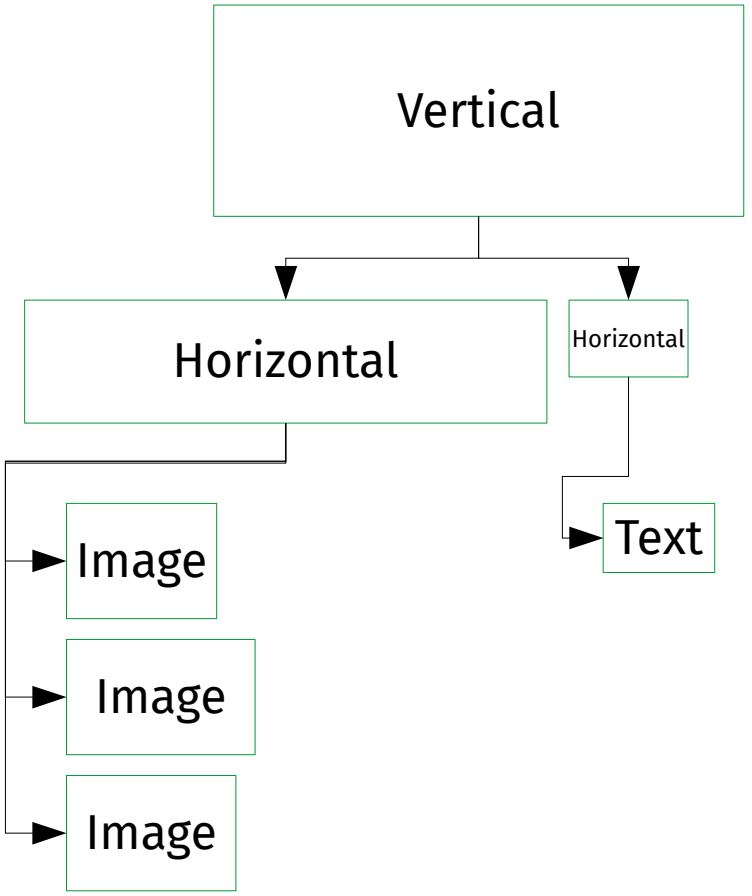


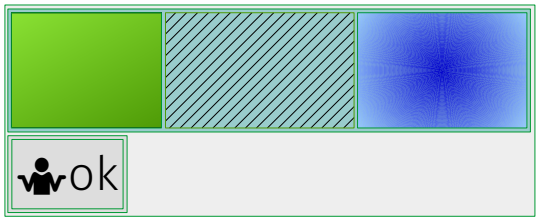
View#measure



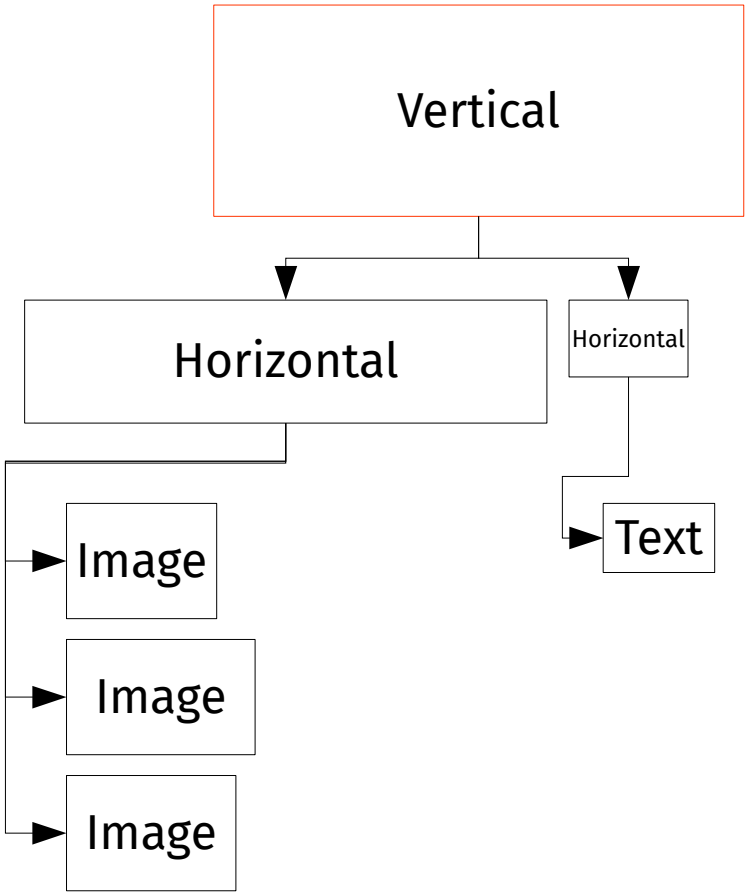


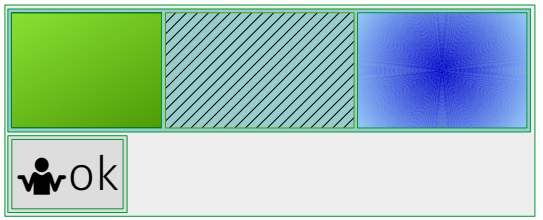
View#measure



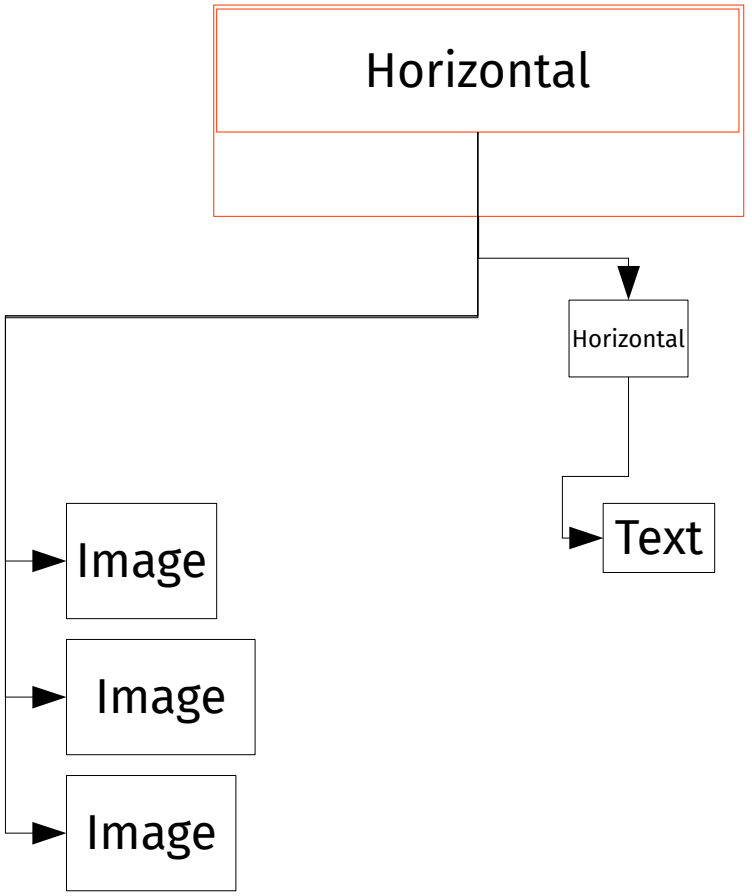


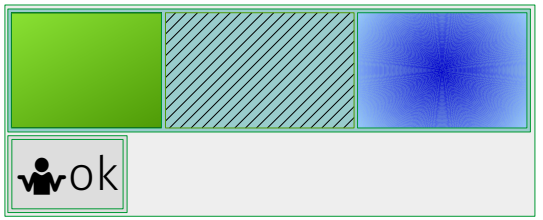
View#layout



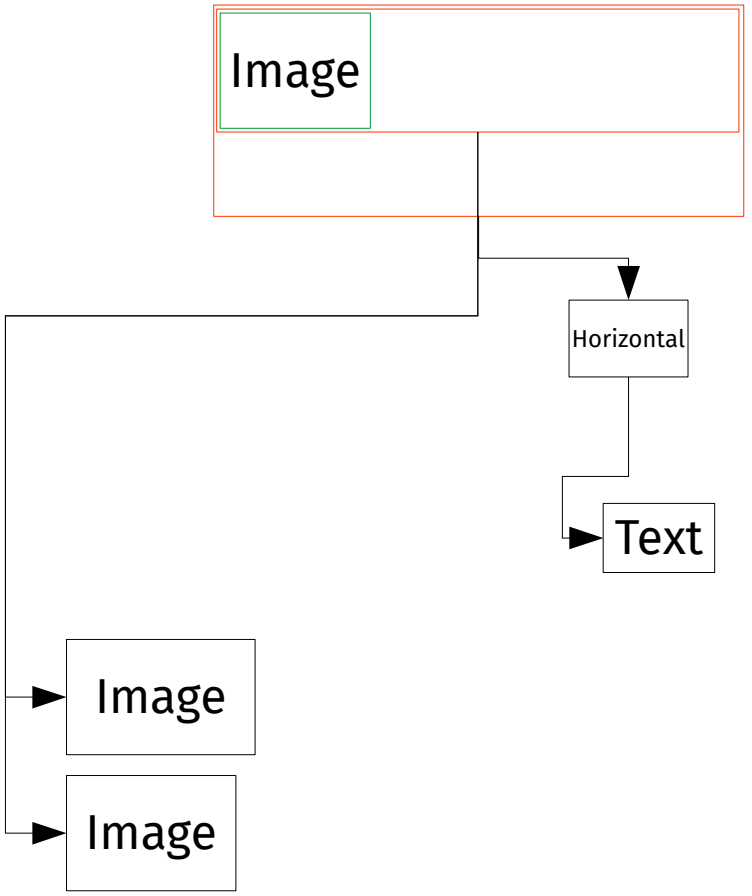


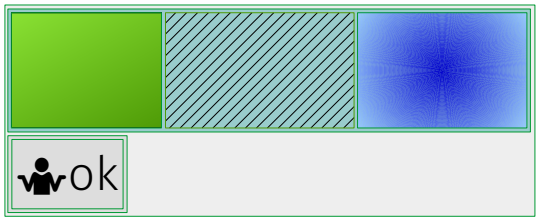
View#layout



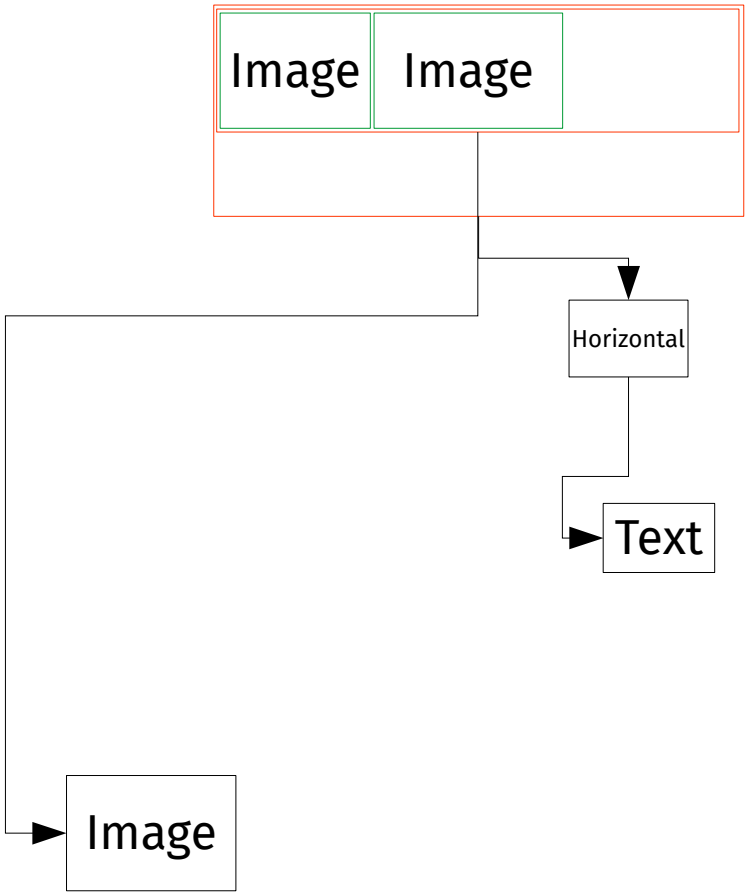


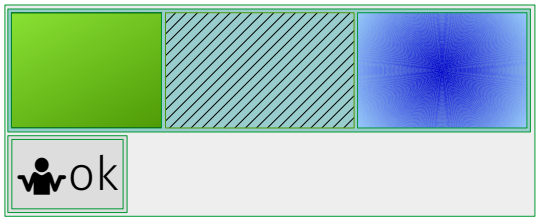
View#layout



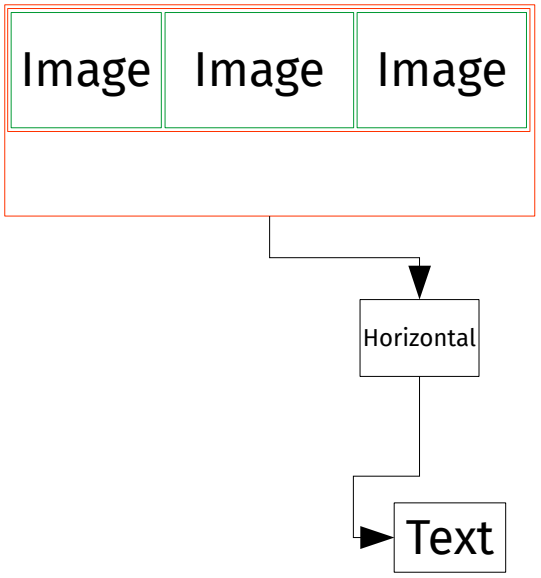


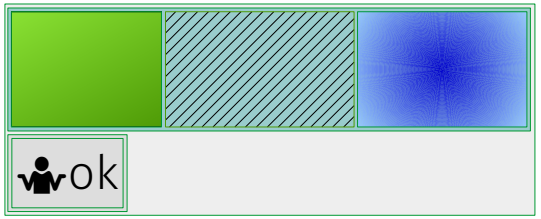
View#layout



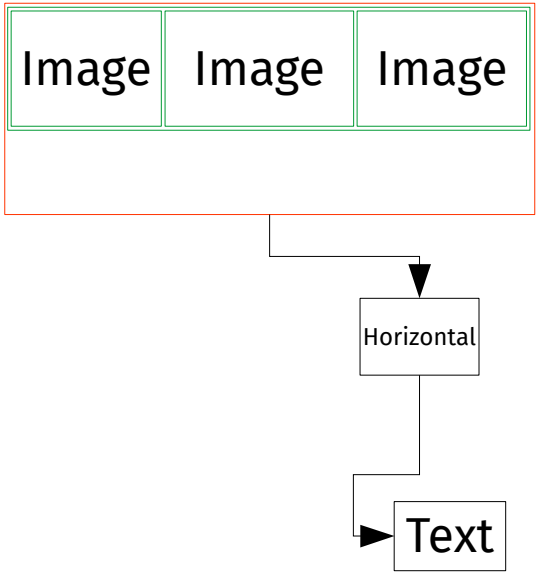


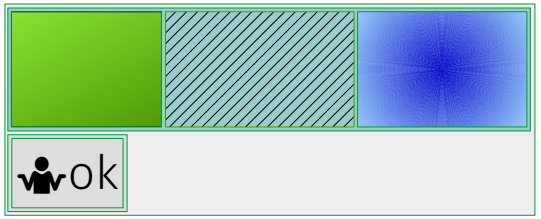
View#layout



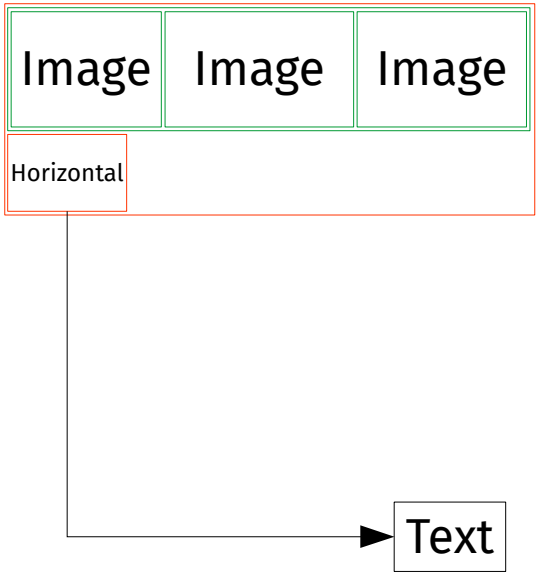


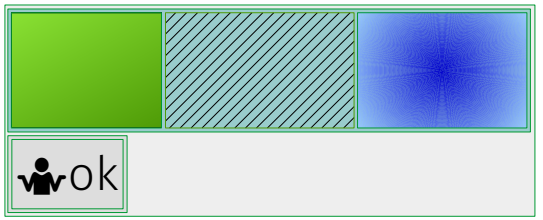
View#layout



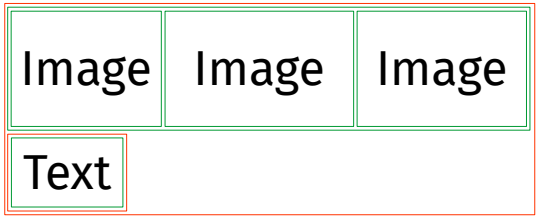


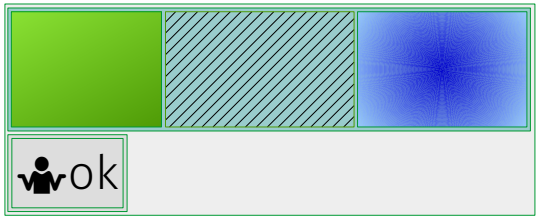
View#layout



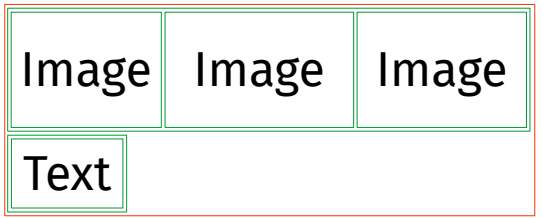


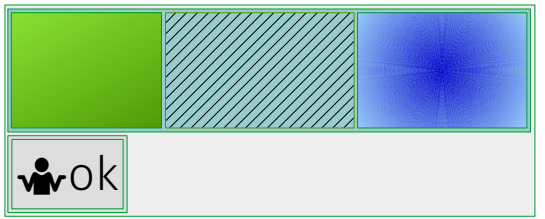
View#layout



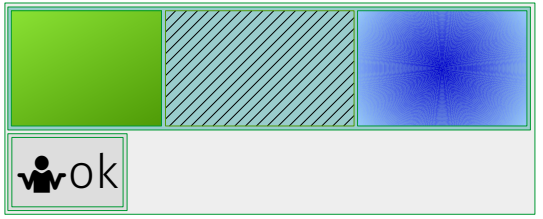


View#layout

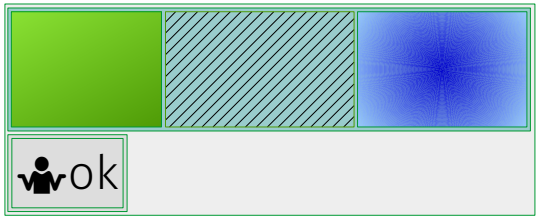




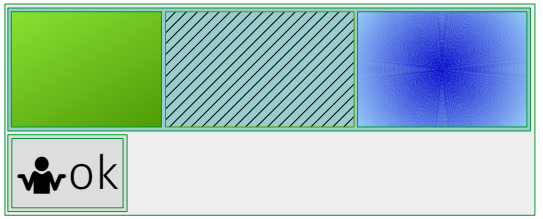
View#layout



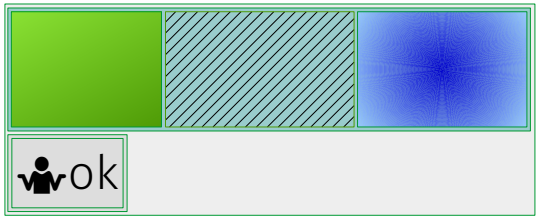
View#draw



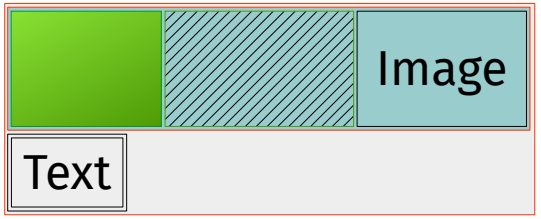
View#draw

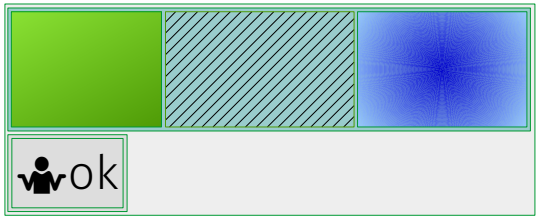


View#draw

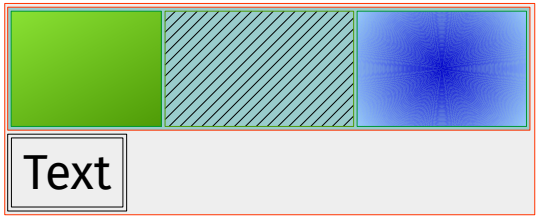


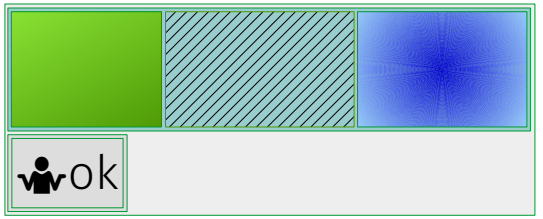
View#draw



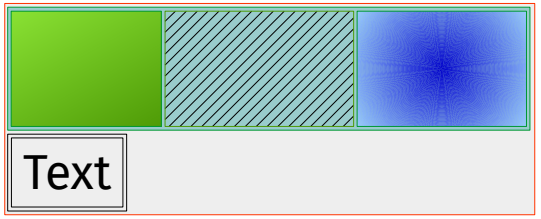


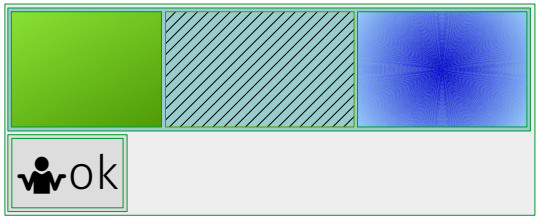
View#draw



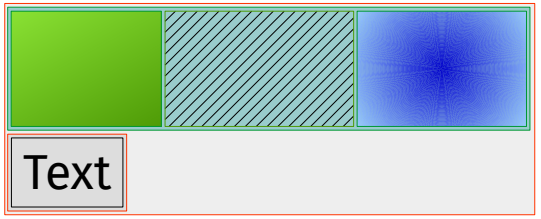


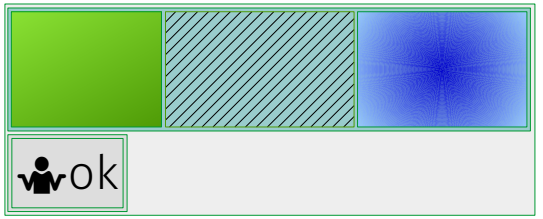
View#draw



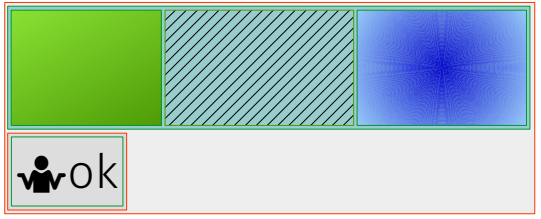


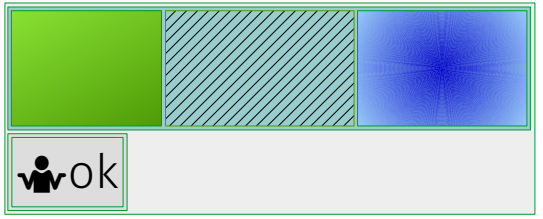
View#draw



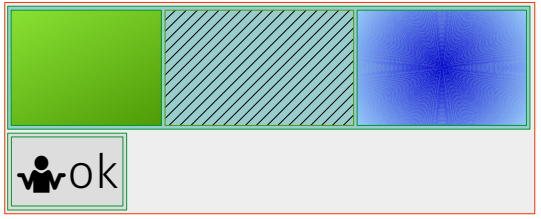


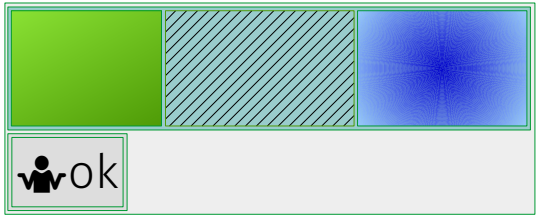
View#draw



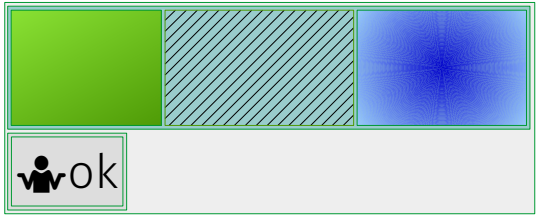


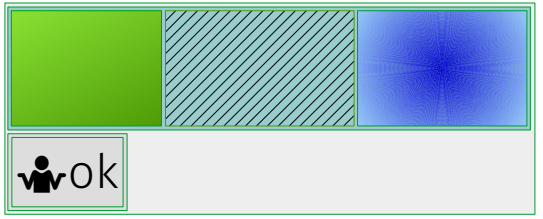
View#draw





View#draw





View passes

- **measure**
- layout
- draw

View#onMeasure

```
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    setMeasuredDimension(
        getDefaultSize(getSuggestedMinimumWidth(), widthMeasureSpec),
        getDefaultSize(getSuggestedMinimumHeight(), heightMeasureSpec)
    );
}
```

View#onMeasure

```
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    setMeasuredDimension(
        getDefaultSize(getSuggestedMinimumWidth(), widthMeasureSpec),
        getDefaultSize(getSuggestedMinimumHeight(), heightMeasureSpec)
    );
}

protected int getSuggestedMinimumWidth() =
    max(mMinWidth, mBackground?.getMinimumWidth() ?: 0)

protected int getSuggestedMinimumHeight() =
    max(mMinHeight, mBackground?.getMinimumHeight() ?: 0)
```

View#onMeasure

```
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    setMeasuredDimension(
        getDefaultSize(getSuggestedMinimumWidth(), widthMeasureSpec),
        getDefaultSize(getSuggestedMinimumHeight(), heightMeasureSpec)
    );
}

public static int getDefaultSize(int size, int measureSpec) =
    when (MeasureSpec.getMode(measureSpec)) {
        MeasureSpec.AT_MOST,
        MeasureSpec.EXACTLY →
            MeasureSpec.getSize(measureSpec)
        else →
            size
    }
```

```
MeasureSpec = (
    mode = EXACTLY | AT_MOST | UNSPECIFIED,
    size = i30,
)
```

```
MeasureSpec = (
    mode = EXACTLY | AT_MOST | UNSPECIFIED,
    size = i30,
)

EXACTLY 100500
```

```
MeasureSpec = (
    mode = EXACTLY | AT_MOST | UNSPECIFIED,
    size = i30,
)

EXACTLY 100500
AT_MOST 1280
```

```
MeasureSpec = (
    mode = EXACTLY | AT_MOST | UNSPECIFIED,
    size = i30,
)

EXACTLY 100500
AT_MOST 1280
UNSPECIFIED 1920
```

```
MeasureSpec = (
    mode = EXACTLY | AT_MOST | UNSPECIFIED,
    size = i30,
)

EXACTLY 100500
AT_MOST 1280
UNSPECIFIED 1920
UNSPECIFIED 0
```

```
MeasureSpec = (
    mode = EXACTLY | AT_MOST | UNSPECIFIED,
    size = i30,
)

EXACTLY 100500
AT_MOST 1280
UNSPECIFIED 1920
UNSPECIFIED 0

mmssssss ssssssss ssssssss ssssssss
```

```
MeasureSpec = (
    mode = EXACTLY | AT_MOST | UNSPECIFIED,
    size = i30,
)

EXACTLY 100500
AT_MOST 1280
UNSPECIFIED 1920
UNSPECIFIED 0

mmssssss ssssssss ssssssss ssssssss
mode = mm000000 00000000 00000000 00000000
size = 00ssssss ssssssss ssssssss ssssssss
```

View#onMeasure

```
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    setMeasuredDimension(
        getDefaultSize(getSuggestedMinimumWidth(), widthMeasureSpec),
        getDefaultSize(getSuggestedMinimumHeight(), heightMeasureSpec)
    );
}

public static int getDefaultSize(int size, int measureSpec) =
    when (MeasureSpec.getMode(measureSpec)) {
        MeasureSpec.AT_MOST,
        MeasureSpec.EXACTLY →
            MeasureSpec.getSize(measureSpec)
        else →
            size
    }
```

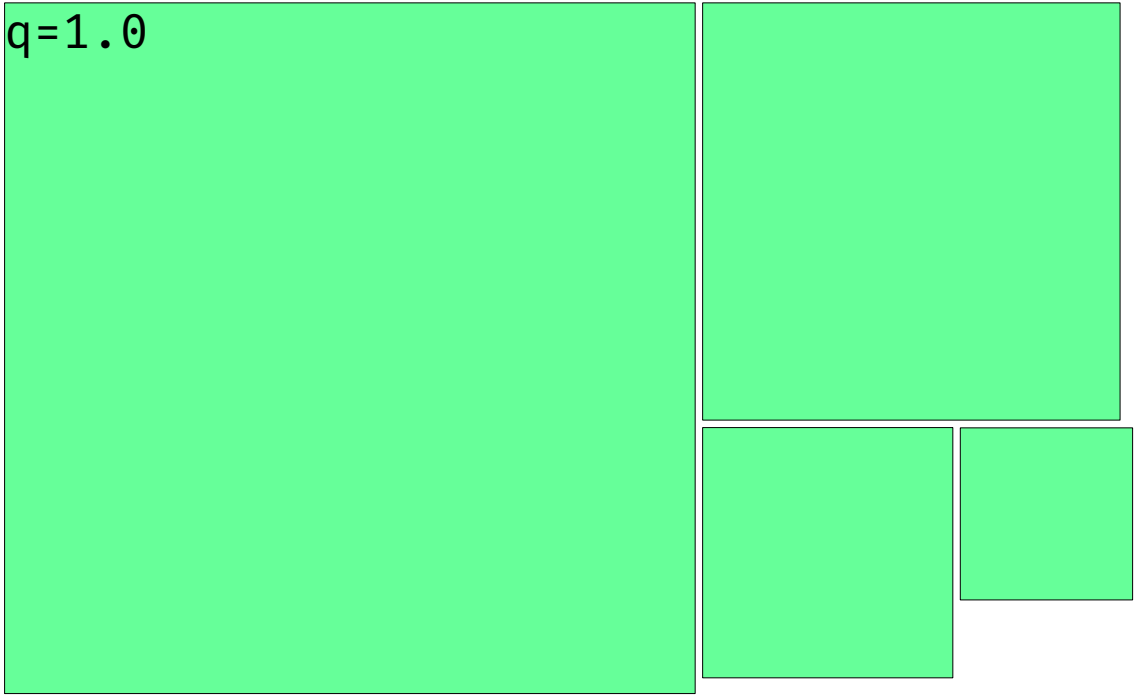
View#onMeasure

```
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    setMeasuredDimension(
        getDefaultSize(getSuggestedMinimumWidth(), widthMeasureSpec),
        getDefaultSize(getSuggestedMinimumHeight(), heightMeasureSpec)
    );
}

public static int getDefaultSize(int size, int measureSpec) =
    when (MeasureSpec.getMode(measureSpec)) {
        MeasureSpec.AT_MOST,
        MeasureSpec.EXACTLY →
            MeasureSpec.getSize(measureSpec)
        else →
            size
    }
```

View#onMeasure

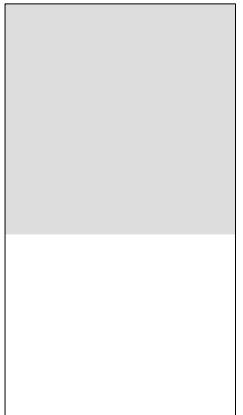
```
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    setMeasuredDimension(
        getDefaultSize(getSuggestedMinimumWidth(), widthMeasureSpec),
        getDefaultSize(getSuggestedMinimumHeight(), heightMeasureSpec)
    );
}
```

AspectFrameLayout



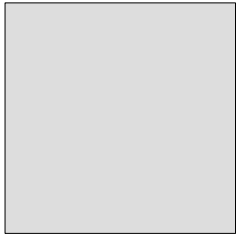
AspectFrameLayout#onMeasure

	EXACTLY w	AT_MOST w	UNSPECIFIED w
EXACTLY h			...
AT_MOST h			
UNSPECIFIED h	...		



AspectFrameLayout#onMeasure

	EXACTLY w	AT_MOST w	UNSPECIFIED w
EXACTLY h			$\frac{h}{q}$×h
AT_MOST h			
UNSPECIFIED h	w×wq		

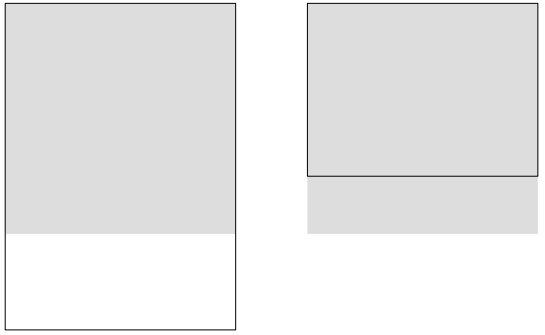


AspectFrameLayout#onMeasure

	EXACTLY w	AT_MOST w	UNSPECIFIED w
EXACTLY h	w×h		$\frac{h}{q}$×h
AT_MOST h			
UNSPECIFIED h	w×wq		

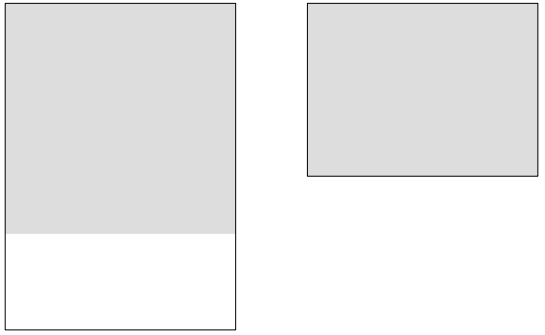
AspectFrameLayout#onMeasure

	EXACTLY w	AT_MOST w	UNSPECIFIED w
EXACTLY h	w×h		$\frac{h}{q}$×h
AT_MOST h	...		
UNSPECIFIED h	w×wq		



AspectFrameLayout#onMeasure

	EXACTLY w	AT_MOST w	UNSPECIFIED w
EXACTLY h	w×h		$\frac{h}{q}$×h
AT_MOST h	...		
UNSPECIFIED h	w×wq		



AspectFrameLayout#onMeasure

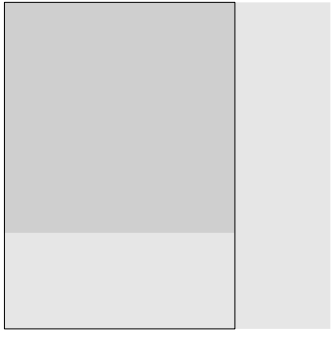
	EXACTLY w	AT_MOST w	UNSPECIFIED w
EXACTLY h	w×h		$\frac{h}{q}$×h
AT_MOST h	w×wq∧h		
UNSPECIFIED h	w×wq		

AspectFrameLayout#onMeasure

	EXACTLY w	AT_MOST w	UNSPECIFIED w
EXACTLY h	w×h	$\frac{h}{q}$∧w×h	$\frac{h}{q}$×h
AT_MOST h	w×wq∧h		
UNSPECIFIED h	w×wq		

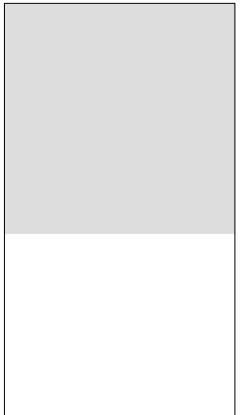
AspectFrameLayout#onMeasure

	EXACTLY w	AT_MOST w	UNSPECIFIED w
EXACTLY h	w×h	$\frac{h}{q}$∧w×h	$\frac{h}{q}$×h
AT_MOST h	w×wq∧h	$\frac{h}{q}$∧w×wq∧h	
UNSPECIFIED h	w×wq		



AspectFrameLayout#onMeasure

	EXACTLY w	AT_MOST w	UNSPECIFIED w
EXACTLY h	w×h	$\frac{h}{q}$∧w×h	$\frac{h}{q}$×h
AT_MOST h	w×wq∧h	$\frac{h}{q}$∧w×wq∧h	$\frac{h}{q}$×h
UNSPECIFIED h	w×wq	w×wq	



AspectFrameLayout#onMeasure

	EXACTLY w	AT_MOST w	UNSPECIFIED w
EXACTLY h	w×h	$\frac{h}{q}$∧w×h	$\frac{h}{q}$×h
AT_MOST h	w×wq∧h	$\frac{h}{q}$∧w×wq∧h	$\frac{h}{q}$×h
UNSPECIFIED h	w×wq	w×wq	w×h

AspectFrameLayout#onMeasure

	EXACTLY w	AT_MOST w	UNSPECIFIED w
EXACTLY h	w×h	$\frac{h}{q}$∧w×h	$\frac{h}{q}$×h
AT_MOST h	w×wq∧h	$\frac{h}{q}$∧w×wq∧h	$\frac{h}{q}$×h
UNSPECIFIED h	w×wq	w×wq	w×h

AspectFrameLayout#onMeasure

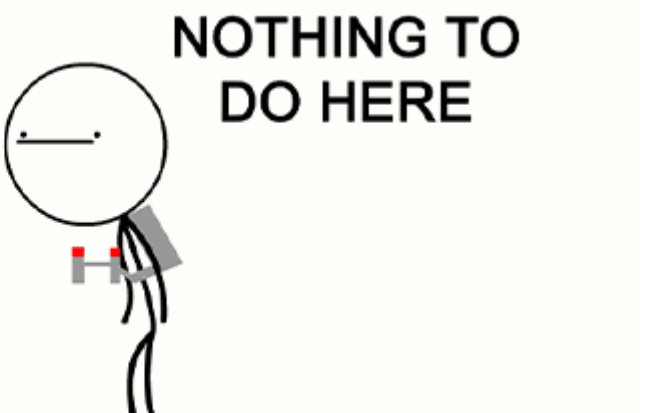
	EXACTLY w	AT_MOST w	UNSPECIFIED w
EXACTLY h	w× h	$\frac{h}{q}$∧w× h	$\frac{h}{q}$×h
AT_MOST h	w×wq∧h	$\frac{h}{q}$∧w×wq∧h	$\frac{h}{q}$×h
UNSPECIFIED h	w× wq	w× wq	w×h

```
private fun aspectMeasureSpec(aMeasureSpec: Int, bMeasureSpec: Int, q: Float): Int {




}
```

AspectFrameLayout#onMeasure

	EXACTLY w	AT_MOST w	UNSPECIFIED w
EXACTLY h	w× h	$\frac{h}{q}$∧w× h	$\frac{h}{q}$×h
AT_MOST h	w×wq∧h	$\frac{h}{q}$∧w×wq∧h	$\frac{h}{q}$×h
UNSPECIFIED h	w× wq	w× wq	w×h

```
private fun aspectMeasureSpec(aMeasureSpec: Int, bMeasureSpec: Int, q: Float): Int {
    val aMode = MeasureSpec.getMode(aMeasureSpec)
    if (aMode == MeasureSpec.EXACTLY) return aMeasureSpec
```




```
}
```

AspectFrameLayout#onMeasure

	EXACTLY w	AT_MOST w	UNSPECIFIED w
EXACTLY h	w× h	$\frac{h}{q}$∧w× h	$\frac{h}{q}$×h
AT_MOST h	w×wq∧h	$\frac{h}{q}$∧w×wq∧h	$\frac{h}{q}$×h
UNSPECIFIED h	w× wq	w× wq	w×h

```
private fun aspectMeasureSpec(aMeasureSpec: Int, bMeasureSpec: Int, q: Float): Int {
    val aMode = MeasureSpec.getMode(aMeasureSpec)
    if (aMode == MeasureSpec.EXACTLY) return aMeasureSpec

    val bMode = MeasureSpec.getMode(bMeasureSpec)
    if (bMode == MeasureSpec.UNSPECIFIED) return aMeasureSpec



}
```

AspectFrameLayout#onMeasure

	EXACTLY w	AT_MOST w	UNSPECIFIED w
EXACTLY h	w× h	$\frac{h}{q}$∧w× h	$\frac{h}{q}$×h
AT_MOST h	w×wq∧h	$\frac{h}{q}$∧w×wq∧h	$\frac{h}{q}$×h
UNSPECIFIED h	w× wq	w× wq	w×h

```
private fun aspectMeasureSpec(aMeasureSpec: Int, bMeasureSpec: Int, q: Float): Int {
    val aMode = MeasureSpec.getMode(aMeasureSpec)
    if (aMode == MeasureSpec.EXACTLY) return aMeasureSpec

    val bMode = MeasureSpec.getMode(bMeasureSpec)
    if (bMode == MeasureSpec.UNSPECIFIED) return aMeasureSpec

    val aspectBSize = (MeasureSpec.getSize(bMeasureSpec) / q).toInt()
    return MeasureSpec.makeMeasureSpec(
        if (aMode == MeasureSpec.AT_MOST) min(aspectBSize, MeasureSpec.getSize(aMeasureSpec))
        else aspectBSize,
        bMode
    )
}
```

AspectFrameLayout#onMeasure

```
override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {
    super.onMeasure(
        aspectMeasureSpec(widthMeasureSpec, heightMeasureSpec, q),
        aspectMeasureSpec(heightMeasureSpec, widthMeasureSpec, 1f / q)
    )









}
```

AspectFrameLayout#onMeasure

	EXACTLY w	AT_MOST w	UNSPECIFIED w
EXACTLY h	w× h	$\frac{h}{q}$∧w× h	$\frac{h}{q}$×h
AT_MOST h	w×wq∧h	$\frac{h}{q}$∧w×wq∧h	$\frac{h}{q}$×h
UNSPECIFIED h	w× wq	w× wq	w×h

AspectFrameLayout#onMeasure

```
override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {
    super.onMeasure(
        aspectMeasureSpec(widthMeasureSpec, heightMeasureSpec, q),
        aspectMeasureSpec(heightMeasureSpec, widthMeasureSpec, 1f / q)
    )

    if (MeasureSpec.getMode(widthMeasureSpec) ≠ MeasureSpec.EXACTLY &&
        MeasureSpec.getMode(heightMeasureSpec) ≠ MeasureSpec.EXACTLY) {
        setMeasuredDimension(
            max(measuredWidth, (measuredHeight / q).toInt()),
            max(measuredHeight, (measuredWidth * q).toInt())
        )
    }
}
```

AspectFrameLayout#onMeasure

```
override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {
    super.onMeasure(
        aspectMeasureSpec(widthMeasureSpec, heightMeasureSpec, q),
        aspectMeasureSpec(heightMeasureSpec, widthMeasureSpec, 1f / q)
    )

    if (MeasureSpec.getMode(widthMeasureSpec) != MeasureSpec.EXACTLY &&
        MeasureSpec.getMode(heightMeasureSpec) != MeasureSpec.EXACTLY) {
        setMeasuredDimension(
            max(measuredWidth, (measuredHeight / q).toInt()),
            max(measuredHeight, (measuredWidth * q).toInt())
        )
    }
}
```

AspectFrameLayout#onMeasure

```
override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {
    super.onMeasure(
        aspectMeasureSpec(widthMeasureSpec, heightMeasureSpec, q),
        aspectMeasureSpec(heightMeasureSpec, widthMeasureSpec, 1f / q)
    )

    if (MeasureSpec.getMode(widthMeasureSpec) != MeasureSpec.EXACTLY &&
        MeasureSpec.getMode(heightMeasureSpec) != MeasureSpec.EXACTLY) {
        setMeasuredDimension(
            max(measuredWidth, (measuredHeight / q).toInt()),
            max(measuredHeight, (measuredWidth * q).toInt())
        )
    }
}
```

AspectFrameLayout

```
override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {
    if (q.isNaN())
        return super.onMeasure(widthMeasureSpec, heightMeasureSpec)

    ...
}
```

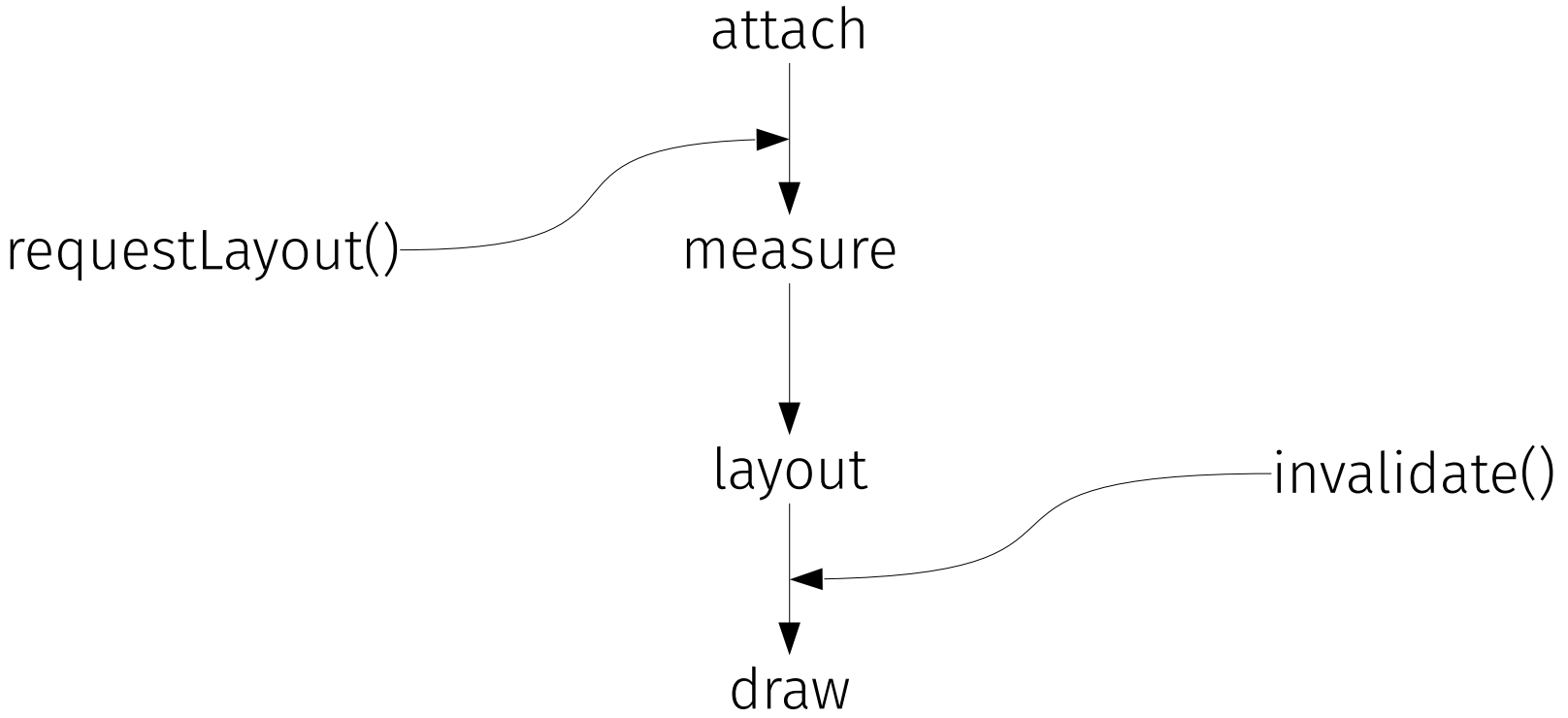
AspectFrameLayout

```
var q: Float = Float.NaN
    set(new) {
        if (field ≠ new) {
            if (new ⩽ 0f || new.isInfinite())
                error { "q must be NaN or ∈(0;∞), $new given" }
            field = new

        }
    }


override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {
    if (q.isNaN())
        return super.onMeasure(widthMeasureSpec, heightMeasureSpec)

    ...
}
```

AspectFrameLayout

```
var q: Float = Float.NaN
    set(new) {
        if (field ≠ new) {
            if (new ⩽ 0f || new.isInfinite())
                error { "q must be NaN or ∈(0;∞), $new given" }
            field = new
            requestLayout()
        }
    }


override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {
    if (q.isNaN())
        return super.onMeasure(widthMeasureSpec, heightMeasureSpec)

    ...
}
```

View lifecycle



measure helpers

```
public class View implements ... {
    public static int resolveSize(int size, int measureSpec)
}

public abstract class ViewGroup extends View implements ... {

    protected void measureChildren(int widthMeasureSpec, int heightMeasureSpec)

    protected void measureChild(
        View child, int parentWidthMeasureSpec, int parentHeightMeasureSpec)

    protected void measureChildWithMargins(
        View child, int parentWidthMeasureSpec, int widthUsed,
        int parentHeightMeasureSpec, int heightUsed)

    public static int getChildMeasureSpec(
        int spec, int padding, int childDimension)
}
```

View passes

- measure
- **layout**
- draw

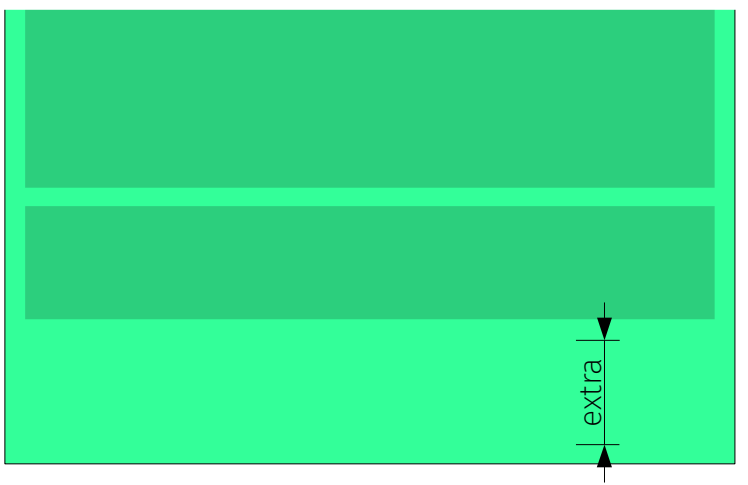
GridLayoutManager#onLayoutChildren

SnapDownLinearLayout

```
override fun onLayout(changed: Boolean,
    left: Int, top: Int, right: Int, bottom: Int) {
    super.onLayout(changed, left, top, right, bottom)




}
```

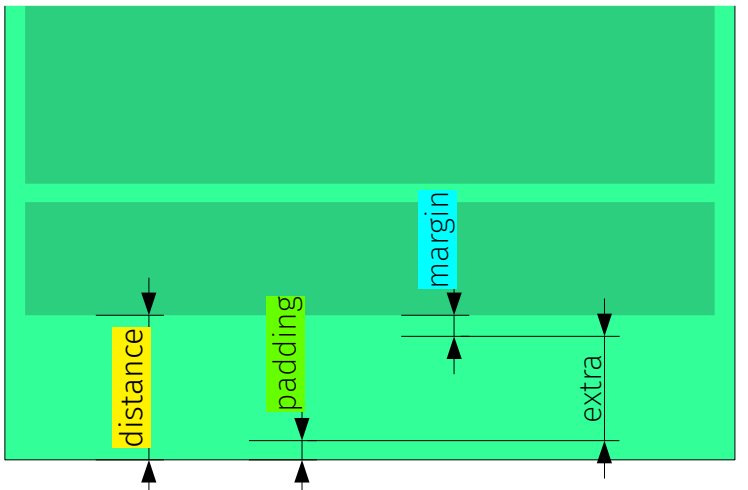
SnapDownLinearLayout

```
override fun onLayout(changed: Boolean,
    left: Int, top: Int, right: Int, bottom: Int) {
    super.onLayout(changed, left, top, right, bottom)

    if (orientation == VERTICAL && childCount > 1) {
        val lastView = getChildAt(childCount - 1)
        val extra = …




    }
}
```



SnapDownLinearLayout

```
override fun onLayout(changed: Boolean,
    left: Int, top: Int, right: Int, bottom: Int) {
    super.onLayout(changed, left, top, right, bottom)

    if (orientation == VERTICAL && childCount > 1) {
        val lastView = getChildAt(childCount - 1)
        val extra = bottom - top - lastView.bottom - paddingBottom -
            ((lastView.layoutParams as? MarginLayoutParams)?.bottomMargin ?: 0)



    }
}
```



SnapDownLinearLayout

```
override fun onLayout(changed: Boolean,
    left: Int, top: Int, right: Int, bottom: Int) {
    super.onLayout(changed, left, top, right, bottom)

    if (orientation == VERTICAL && childCount > 1) {
        val lastView = getChildAt(childCount - 1)
        val extra = bottom - top - lastView.bottom - paddingBottom -
            ((lastView.layoutParams as? MarginLayoutParams)?.bottomMargin ?: 0)
        if (extra > 0) {
            lastView.offsetTopAndBottom(extra)
        }
    }
}
```

View passes

- measure
- layout
- draw

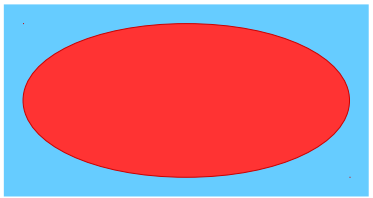
View drawing

```
@CallSuper
public void draw(Canvas canvas) {
    ...
}

protected void onDraw(Canvas canvas) {
}

protected void dispatchDraw(Canvas canvas) {
}
```

Drawables

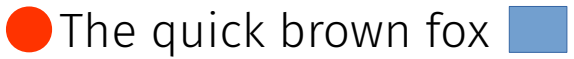
```
android:background="@drawable/ ... "
android:background="@color/ ... "
android:background="#00ff66"
setBackground(Drawable)
setBackgroundColor(@ColorInt)
setBackgroundResource(@DrawableRes)
```

Drawables



```
android:src="@drawable/ ... "
setImageDrawable(Drawable)
setImageResource(@DrawableRes)
setImageBitmap(Bitmap)
```

Drawables




```
android:drawableLeft="@drawable/ ... "
android:drawableRight="@drawable/ ... "
setCompoundDrawables( ... )
setCompoundDrawablesRelative( ... )
setCompoundDrawablesWithIntrinsicBounds( ... )
setCompoundDrawablesRelativeWithIntrinsicBounds( ... )
```

SDK Drawables

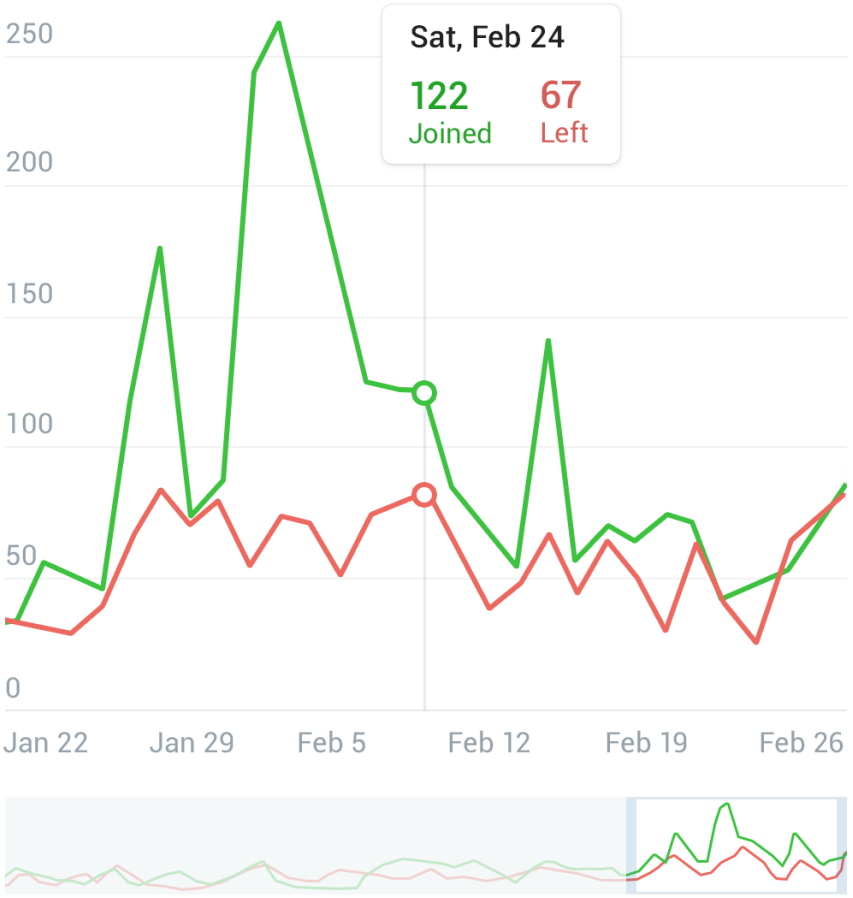
GradientDrawable

```xml
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <solid android:color="#3399FF" />
    <corners android:radius="42dp" />
</shape>
```

```
fun RoundRectDrawable(color: Int, corners: Float): Drawable =
    GradientDrawable().apply {
        setColor(color)
        cornerRadius = corners
    }
```

Ripple+StateListDrawable

```
fun Context.ClickableRoundRectDrawable(): Drawable =
    RippleDrawable(
        ColorStateList.valueOf(0x20_000000),
        StateListDrawable().apply {
            addState(
                intArrayOf(android.R.attr.state_enabled),
                RoundRectDrawable(0xFF_3399FF.toInt(), dp(42).toFloat())
            )
            addState(
                intArrayOf(-android.R.attr.state_enabled),
                RoundRectDrawable(0xFF_E6E9EB.toInt(), dp(42).toFloat())
            )
        },
        null
    )
```



Joined

Left

https://t.me/contest/10

```java
public abstract class Drawable {

    /**
     * Draw in its bounds (set via setBounds) respecting optional effects such
     * as alpha (set via setAlpha) and color filter (set via setColorFilter).
     *
     * @param canvas The canvas to draw into
     */
    public abstract void draw(
        @NonNull Canvas canvas
    );

    public abstract void setAlpha(
        @IntRange(from=0, to=255) int alpha
    );

    public abstract void setColorFilter(
        @Nullable ColorFilter colorFilter
    );

    public abstract int getOpacity();
}
```

```
class DiagonalLineDrawable : Drawable() {




}
```

```
class DiagonalLineDrawable : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG)
        .also { it.color = Color.BLACK }









}
```

```
class DiagonalLineDrawable : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG)
        .also { it.color = Color.BLACK }
    override fun draw(canvas: Canvas) {
        val bnds: Rect = bounds



    }




}
```

```
class DiagonalLineDrawable : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG)
        .also { it.color = Color.BLACK }
    override fun draw(canvas: Canvas) {
        val bnds: Rect = bounds
        canvas.drawLine(
            bnds.left.toFloat(), bnds.top.toFloat(),
            bnds.right.toFloat(), bnds.bottom.toFloat(),
            paint
        )
    }









}
```

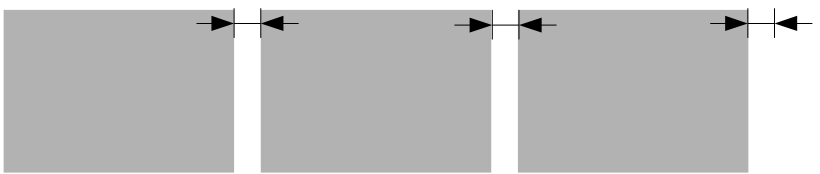
```
class DiagonalLineDrawable : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG)
        .also { it.color = Color.BLACK } // 0xAARRGGBB
    override fun draw(canvas: Canvas) {
        val bnds: Rect = bounds
        canvas.drawLine(
            bnds.left.toFloat(), bnds.top.toFloat(),
            bnds.right.toFloat(), bnds.bottom.toFloat(),
            paint
        )
    }
    override fun setAlpha(alpha: Int) {
        paint.alpha = alpha
    }



}
```

```
class DiagonalLineDrawable : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG)
        .also { it.color = Color.BLACK }
    override fun draw(canvas: Canvas) {
        val bnds: Rect = bounds
        canvas.drawLine(
            bnds.left.toFloat(), bnds.top.toFloat(),
            bnds.right.toFloat(), bnds.bottom.toFloat(),
            paint
        )
    }
    override fun setAlpha(alpha: Int) {
        paint.alpha = alpha
    }
    override fun setColorFilter(colorFilter: ColorFilter?) {
        paint.colorFilter = colorFilter
    }


}
```

```
class DiagonalLineDrawable : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG)
        .also { it.color = Color.BLACK }
    override fun draw(canvas: Canvas) {
        val bnds: Rect = bounds
        canvas.drawLine(
            bnds.left.toFloat(), bnds.top.toFloat(),
            bnds.right.toFloat(), bnds.bottom.toFloat(),
            paint
        )
    }
    override fun setAlpha(alpha: Int) {
        paint.alpha = alpha
    }
    override fun setColorFilter(colorFilter: ColorFilter?) {
        paint.colorFilter = colorFilter
    }
    override fun getOpacity(): Int =
        PixelFormat.TRANSLUCENT
}
```

Drawable dimensions

```
recyclerView {
  ...
  addItemDecoration(
    DividerItemDecoration(context, LinearLayoutManager.HORIZONTAL)
      .also { it.setDrawable(SpaceDrawable(dp(10), 0)) }
  )
```



Drawable dimensions

```
horizontalLayout {
  dividerDrawable = SpaceDrawable(dip(10), 0)
  showDividers = LinearLayout.SHOW_DIVIDER_MIDDLE
```

Drawable dimensions

```
fun SpaceDrawable(@Px width: Int, @Px height: Int): Drawable =
  object : ColorDrawable() {
    override fun getIntrinsicWidth(): Int = width
    override fun getIntrinsicHeight(): Int = height
  }
```

```
class RoundedBitmapDrawable(


) : Drawable() {
```


```
}
```

```
class RoundedBitmapDrawable(

    private val radius: Float,
) : Drawable() {




    override fun draw(canvas: Canvas) {
        val bnds = bounds
        canvas.drawRoundRect(
            bnds.left.toFloat(), bnds.top.toFloat(),
            bnds.right.toFloat(), bnds.bottom.toFloat(),
            radius, radius,
            paint
        )
    }




}
```

```
class RoundedBitmapDrawable(
    private val bitmap: Bitmap,
    private val radius: Float,
) : Drawable() {

    private val paint = Paint(Paint.ANTI_ALIAS_FLAG).also {
        it.shader = BitmapShader(bitmap, Shader.TileMode.CLAMP, Shader.TileMode.CLAMP)
    }
    override fun draw(canvas: Canvas) {
        val bnds = bounds
        canvas.drawRoundRect(
            bnds.left.toFloat(), bnds.top.toFloat(),
            bnds.right.toFloat(), bnds.bottom.toFloat(),
            radius, radius,
            paint
        )
    }

}
```

romainguy

```
class RoundedBitmapDrawable(
    private val bitmap: Bitmap,
    private val radius: Float,
) : Drawable() {

    private val paint = Paint(Paint.ANTI_ALIAS_FLAG).also {
        it.shader = BitmapShader(bitmap, Shader.TileMode.CLAMP, Shader.TileMode.CLAMP)
    }
    override fun draw(canvas: Canvas) {
        val bnds = bounds
        canvas.drawRoundRect(
            bnds.left.toFloat(), bnds.top.toFloat(),
            bnds.right.toFloat(), bnds.bottom.toFloat(),
            radius, radius,
            paint
        )
    }
    override fun getIntrinsicWidth(): Int = bitmap.width
    override fun getIntrinsicHeight(): Int = bitmap.height



}
```

```
class RoundedBitmapDrawable(
    private val bitmap: Bitmap,
    private val radius: Float,
) : Drawable() {

    private val paint = Paint(Paint.ANTI_ALIAS_FLAG).also {
        it.shader = BitmapShader(bitmap, Shader.TileMode.CLAMP, Shader.TileMode.CLAMP)
    }
    override fun draw(canvas: Canvas) {
        val bnds = bounds
        canvas.drawRoundRect(
            bnds.left.toFloat(), bnds.top.toFloat(),
            bnds.right.toFloat(), bnds.bottom.toFloat(),
            radius, radius,
            paint
        )
    }
    override fun getIntrinsicWidth(): Int = bitmap.width
    override fun getIntrinsicHeight(): Int = bitmap.height

    override fun setAlpha(alpha: Int) { ... }
    override fun setColorFilter(colorFilter: ColorFilter?) { ... }
    override fun getOpacity(): Int = ...
}
```

Drawable invalidation

```
class RoundedBitmapDrawable(
    private val bitmap: Bitmap,
    radius: Float,
) : Drawable() {

    var radius: Float = radius
        set(new) {
            if (new ≠ field) {
                require(new ⩾ 0 && new ≠ Float.POSITIVE_INFINITY)
                field = new

            }
        }
```

Drawable invalidation

```
class RoundedBitmapDrawable(
    private val bitmap: Bitmap,
    radius: Float,
) : Drawable() {

    var radius: Float = radius
        set(new) {
            if (new ≠ field) {
                require(new ≥ 0 && new ≠ Float.POSITIVE_INFINITY)
                field = new
                invalidateSelf()
            }
        }
```

Canvas

drawArc

drawTextOnPath

drawColor

drawPoint

drawRGB

drawRoundRect

drawRect

drawOval

drawPath

drawPatch

drawLine

drawText

drawCircle

drawARGB

drawBitmap

Canvas

save, restore

clipPath

clipRect

translate

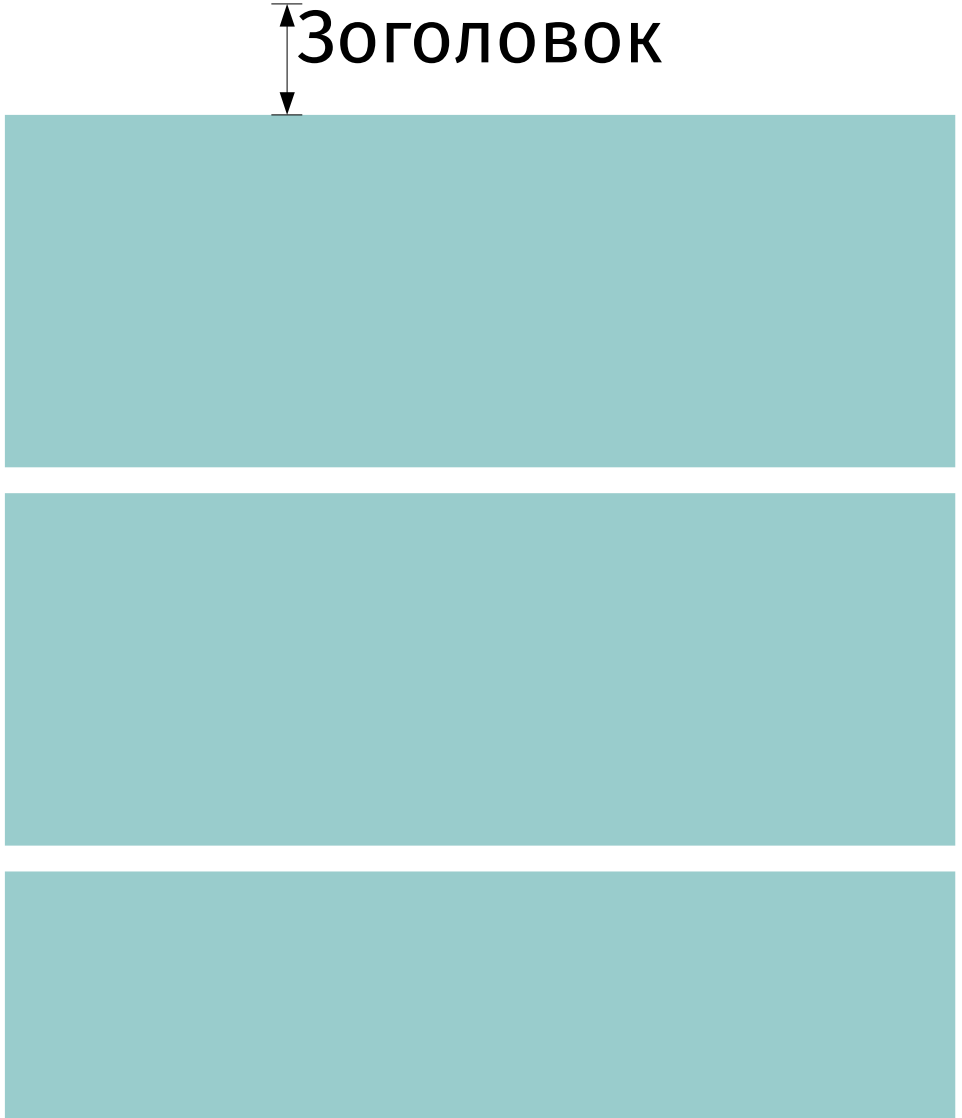
skew

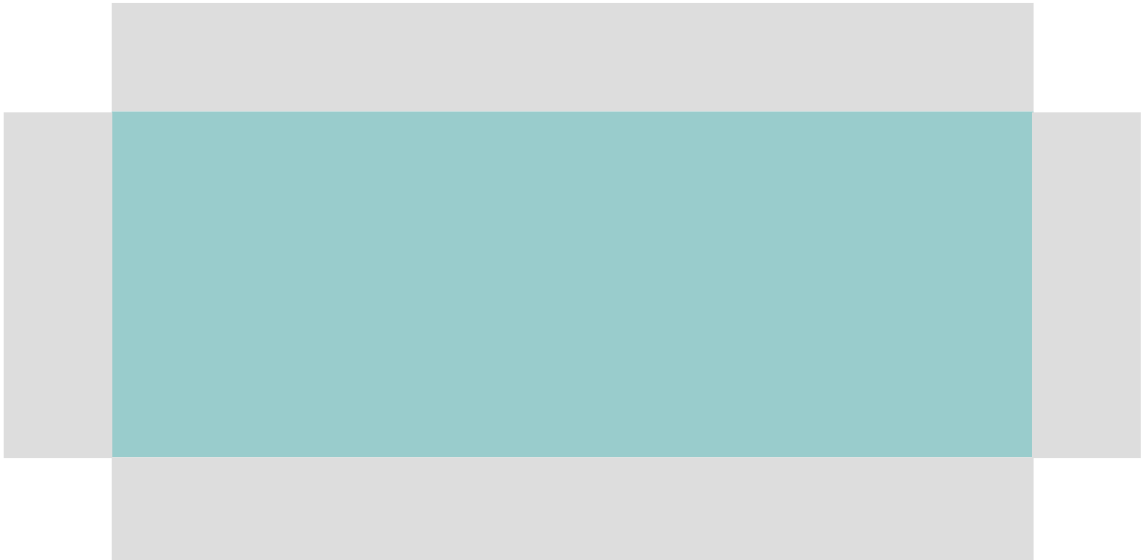
scale

rotate

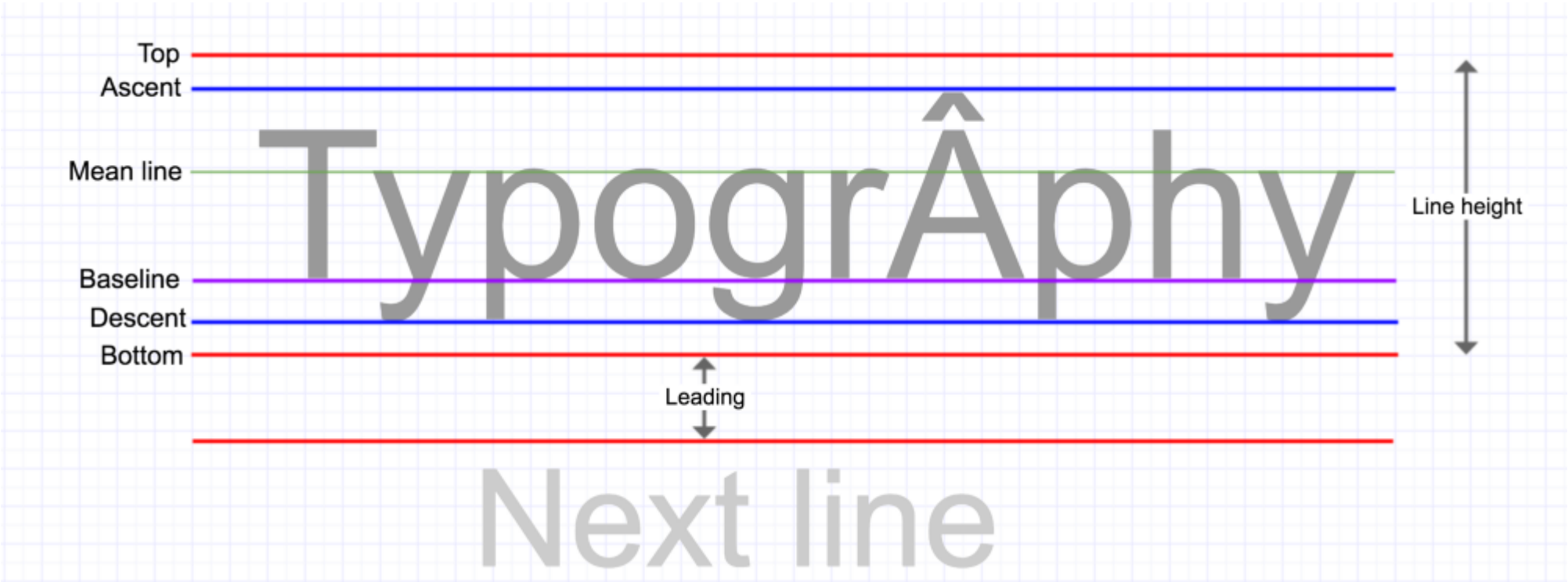
ItemDecoration

```
public abstract static class ItemDecoration {

    public void getItemOffsets(
        @NonNull Rect outRect, @NonNull View view,
        @NonNull RecyclerView parent, @NonNull State state
    )

    public void onDraw(
        @NonNull Canvas c, @NonNull RecyclerView parent, @NonNull State state
    )

    public void onDrawOver(
        @NonNull Canvas c, @NonNull RecyclerView parent, @NonNull State state
    )

}
```


Зоголовок

```
object : RecyclerView.ItemDecoration() {
    private val paint = TextPaint(Paint.ANTI_ALIAS_FLAG).also { it.color = Color.BLACK }
    private var spacing: Int = 0
    private fun measure(parent: RecyclerView) {
        spacing = parent.dip(10)
        paint.textSize = parent.sp(24f)
    }

    override fun getItemOffsets(
        outRect: Rect, view: View, parent: RecyclerView, state: RecyclerView.State) {
        if (spacing == 0) measure(parent)



    }



}
```

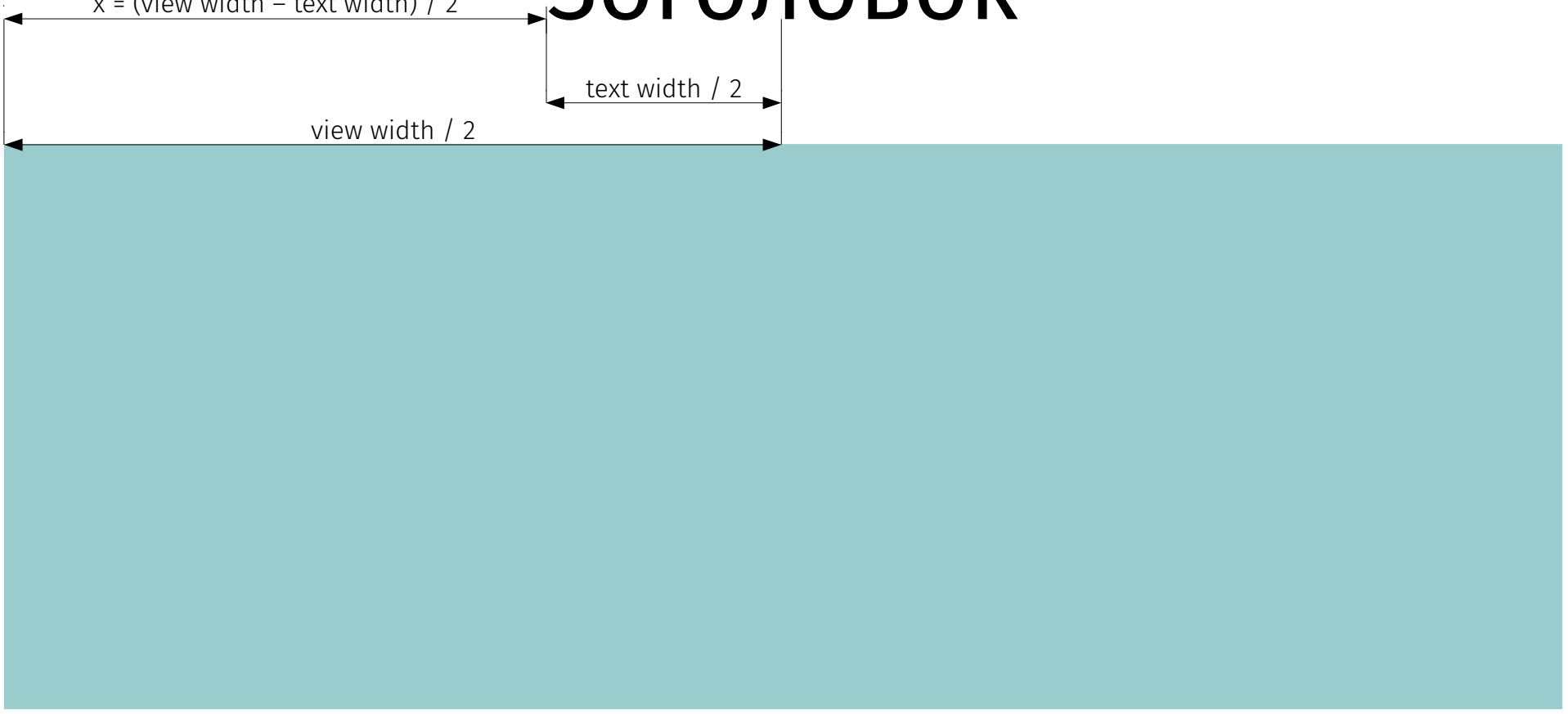
FontMetrics



Zoголовок

```
object : RecyclerView.ItemDecoration() {
    private val paint = TextPaint(Paint.ANTI_ALIAS_FLAG).also { it.color = Color.BLACK }
    private var spacing: Int = 0
    private fun measure(parent: RecyclerView) {
        spacing = parent.dip(10)
        paint.textSize = parent.sp(24f)
    }

    override fun getItemOffsets(
        outRect: Rect, view: View, parent: RecyclerView, state: RecyclerView.State) {
        if (spacing == 0) measure(parent)

        val top =
            if (parent.getChildViewHolder(view).adapterPosition == 0)
                paint.fontMetricsInt.run { bottom - top } + spacing
            else 0
        outRect.set(0, top, 0, 0)
    }



}
```

Заголовок



```
object : RecyclerView.ItemDecoration() {
    private val paint = TextPaint(Paint.ANTI_ALIAS_FLAG).also { it.color = Color.BLACK }
    private var spacing: Int = 0
    private fun measure(parent: RecyclerView) {
        spacing = parent.dip(10)
        paint.textSize = parent.sp(24f)
    }

    override fun getItemOffsets(
        outRect: Rect, view: View, parent: RecyclerView, state: RecyclerView.State) {
        if (spacing == 0) measure(parent)

        val top =
            if (parent.getChildViewHolder(view).adapterPosition == 0)
                paint.fontMetricsInt.run { bottom - top } + spacing
            else 0
        outRect.set(0, top, 0, 0)
    }

    override fun onDraw(c: Canvas, parent: RecyclerView, state: RecyclerView.State) {
        parent.findViewHolderForAdapterPosition(0)?.itemView?.let { view →
            val x = view.x + (view.width - paint.measureText(myOrdText)) / 2
            val y = view.y - paint.fontMetricsInt.bottom - spacing
            c.drawText("Заголовок", x, y.toFloat(), paint)
        }
    }
}
```

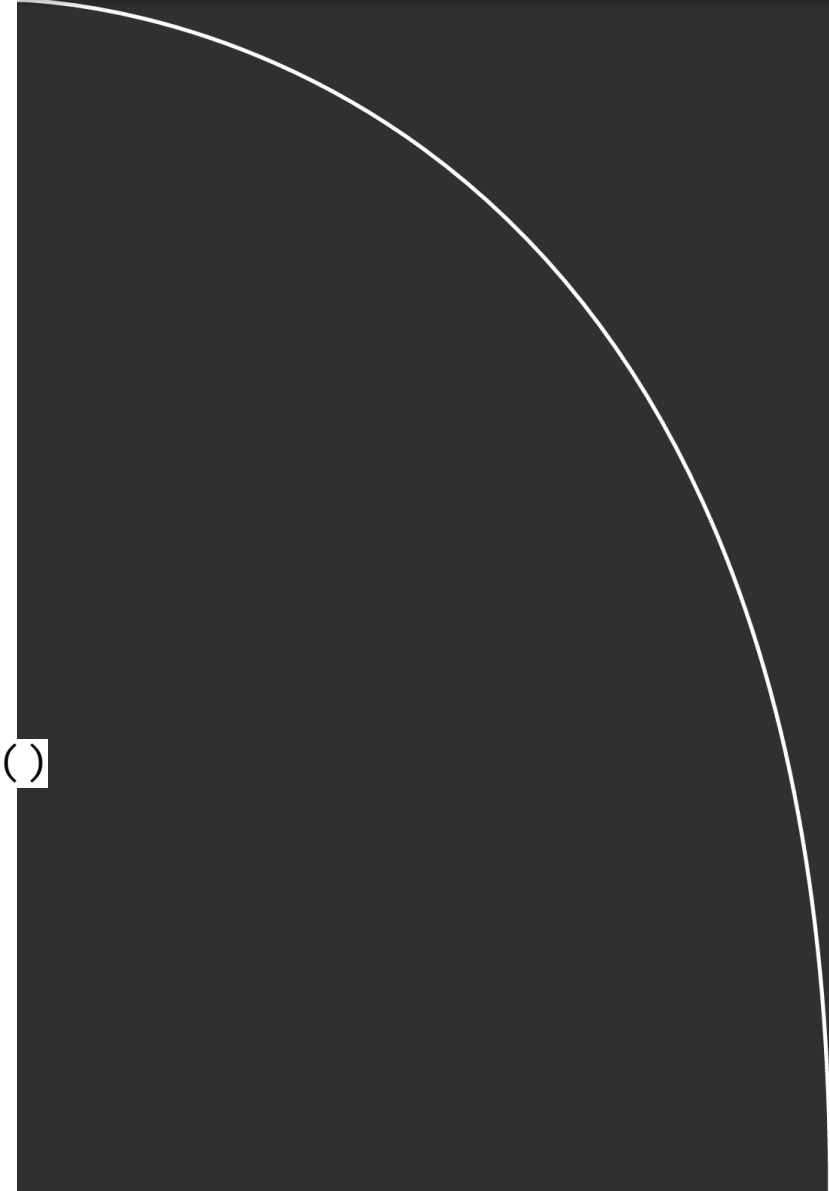
```
class CubicDrawable : Drawable() {



    private val path = Path()









}
```

```
class CubicDrawable(context: Context) : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG)
        .also {
            it.style = Paint.Style.STROKE
            it.color = Color.WHITE
            it.strokeWidth = context.dp(2)
        }
    private val path = Path()
```



```
}
```

```
class CubicDrawable(context: Context) : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG)
        .also {
            it.style = Paint.Style.STROKE
            it.color = Color.WHITE
            it.strokeWidth = context.dp(2)
        }
    private val path = Path()

    override fun draw(canvas: Canvas) {
        path.rewind()
        val bnds = bounds
        path.cubicTo(
            bnds.left.toFloat(), bnds.top.toFloat(),
            bnds.right.toFloat(), bnds.top.toFloat(),
            bnds.right.toFloat(), bnds.bottom.toFloat()
        )
        canvas.drawPath(path, paint)
    }



}
```

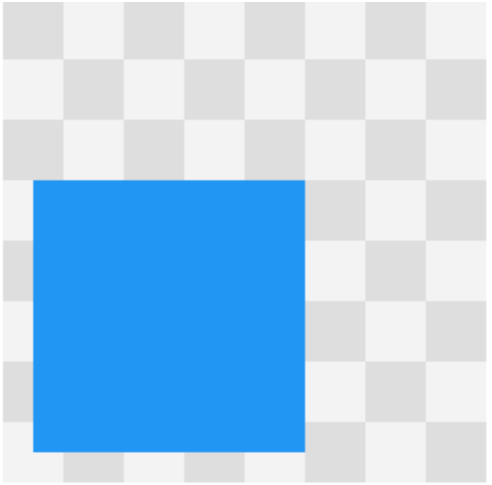


```
class CubicDrawable(context: Context) : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG)
        .also {
            it.style = Paint.Style.FILL
            it.color = Color.WHITE
            it.strokeWidth = context.dp(2)
        }
    private val path = Path()

    override fun draw(canvas: Canvas) {
        path.rewind()
        val bnds = bounds
        path.cubicTo(
            bnds.left.toFloat(), bnds.top.toFloat(),
            bnds.right.toFloat(), bnds.top.toFloat(),
            bnds.right.toFloat(), bnds.bottom.toFloat()
        )
        canvas.drawPath(path, paint)
    }



}
```

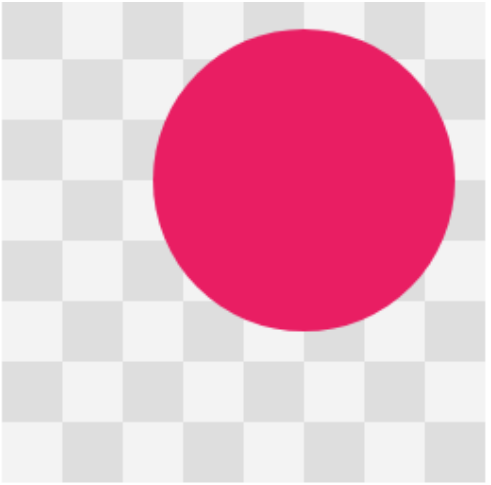
```
class CubicDrawable(context: Context) : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG)
        .also {
            it.style = Paint.Style.STROKE
            it.color = Color.WHITE
            it.strokeWidth = context.dp(2)
        }
    private val path = Path()
    override fun onBoundsChange(bnds: Rect) {
        path.rewind()
        path.cubicTo(
            bnds.left.toFloat(), bnds.top.toFloat(),
            bnds.right.toFloat(), bnds.top.toFloat(),
            bnds.right.toFloat(), bnds.bottom.toFloat()
        )
    }
    override fun draw(canvas: Canvas) {
        canvas.drawPath(path, paint)
    }



}
```

```
class CubicDrawable(context: Context) : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG)
        .also {
            it.style = Paint.Style.STROKE
            it.color = Color.WHITE
            it.strokeWidth = context.dp(2)
        }
    private val path = Path()
    override fun onBoundsChange(bnds: Rect) {
        path.rewind()
        path.cubicTo(
            bnds.left.toFloat(), bnds.top.toFloat(),
            bnds.right.toFloat(), bnds.top.toFloat(),
            bnds.right.toFloat(), bnds.bottom.toFloat()
        )
    }
    override fun draw(canvas: Canvas) {
        canvas.drawPath(path, paint)
    }
    override fun setAlpha(alpha: Int) { ... }
    override fun setColorFilter(colorFilter: ColorFilter?) { ... }
    override fun getOpacity(): Int = ...
}
```

PorterDuff



Source image

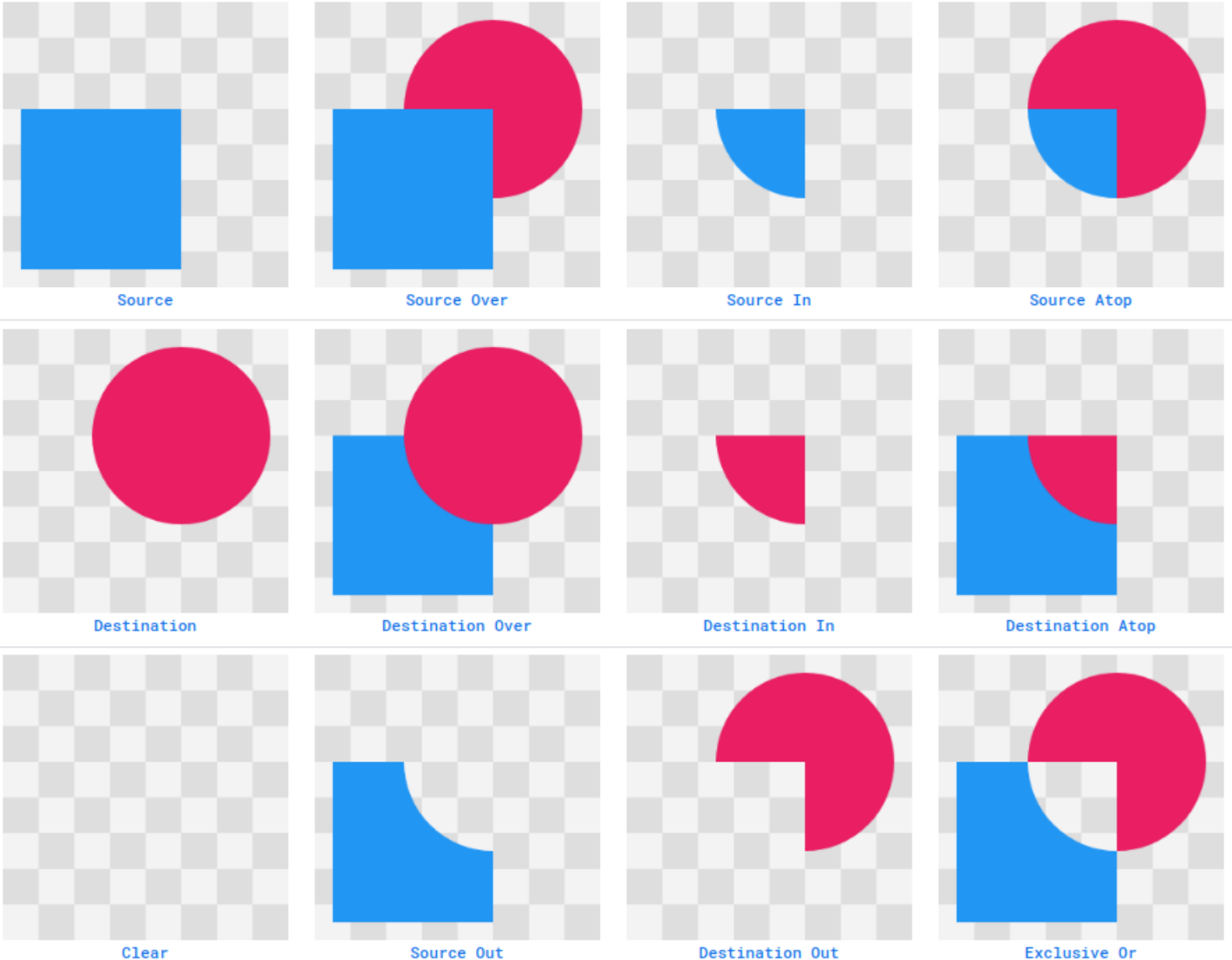


Destination image

The order of drawing operations used to generate each diagram is shown in the following code snippet:

```
Paint paint = new Paint();
canvas.drawBitmap(destinationImage, 0, 0, paint);

PorterDuff.Mode mode = // choose a mode
paint.setXfermode(new PorterDuffXfermode(mode));

canvas.drawBitmap(sourceImage, 0, 0, paint);
```

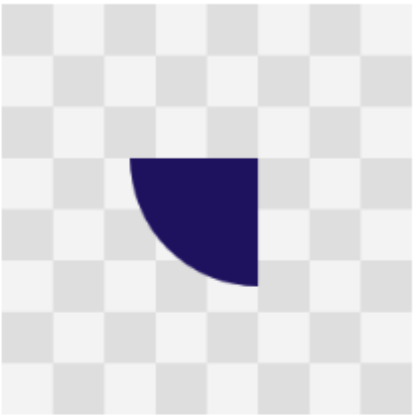
Alpha compositing modes



Blending modes

Darken

Lighten



Multiply

Screen

Overlay

```
class PorterDuffTextDrawable(



) : Drawable() {









}
```

```
class PorterDuffTextDrawable(
    private val background: Shader,



) : Drawable() {
    private val bgPaint = Paint().also { it.shader = background }



    override fun draw(canvas: Canvas) {
        canvas.drawRect(bounds, bgPaint)









    }


}
```

```
class PorterDuffTextDrawable(
    private val background: Shader,
    @ColorInt textColor: Int,
    private val text: String,
    mode: PorterDuff.Mode
) : Drawable() {
    private val bgPaint = Paint().also { it.shader = background }
    private val textPaint = TextPaint(Paint.ANTI_ALIAS_FLAG).also {
        it.color = textColor
        it.xfermode = PorterDuffXfermode(mode)
    }
    override fun draw(canvas: Canvas) {
        canvas.drawRect(bounds, bgPaint)

        textPaint.textSize = .8f * bounds.height()
        canvas.drawText(
            text,
            bounds.left + bounds.width()/2 - textPaint.measureText(text)/2,
            bounds.bottom - textPaint.fontMetrics.bottom,
            textPaint
        )
    }


}
```

```
class PorterDuffTextDrawable(
    private val background: Shader,
    @ColorInt textColor: Int,
    private val text: String,
    mode: PorterDuff.Mode
) : Drawable() {
    private val bgPaint = Paint().also { it.shader = background }
    private val textPaint = TextPaint(Paint.ANTI_ALIAS_FLAG).also {
        it.color = textColor
        it.xfermode = PorterDuffXfermode(mode)
    }
    override fun draw(canvas: Canvas) {
        canvas.drawRect(bounds, bgPaint)

        textPaint.textSize = .8f * bounds.height()
        canvas.drawText(
            text,
            bounds.left + bounds.width()/2 - textPaint.measureText(text)/2,
            bounds.bottom - textPaint.fontMetrics.bottom,
            textPaint
        )
    }
    override fun setAlpha(alpha: Int) { ... }
    override fun setColorFilter(colorFilter: ColorFilter?) { ... }
    override fun getOpacity(): Int = ...
}
```

```
setContentView(View(this).also {
    it.background = PorterDuffTextDrawable(
        LinearGradient(
            0f, 0f, 0f, dp(64),
            intArrayOf(Color.GREEN, Color.YELLOW, Color.RED),
            null,
            Shader.TileMode.CLAMP
        ),



    )

}, FrameLayout.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT, dip(64)))
```


```
setContentView(View(this).also {
    it.background = PorterDuffTextDrawable(
        LinearGradient(
            0f, 0f, 0f, dp(64),
            intArrayOf(Color.GREEN, Color.YELLOW, Color.RED),
            null,
            Shader.TileMode.CLAMP
        ),
        0xFF_3399FF.toInt(),
        "Превед!",
        PorterDuff.Mode.DARKEN
    )

}, FrameLayout.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT, dip(64)))
```


```
setContentView(View(this).also {
    it.background = PorterDuffTextDrawable(
        LinearGradient(
            0f, 0f, 0f, dp(64),
            intArrayOf(Color.GREEN, Color.YELLOW, Color.RED),
            null,
            Shader.TileMode.CLAMP
        ),
        0xFF_3399FF.toInt(),
        "Превед!",
        PorterDuff.Mode.DARKEN
    )
    it.setLayerType(View.LAYER_TYPE_SOFTWARE, null)
}, FrameLayout.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT, dip(64)))
```

Finally, we need to update our view to use the software layer since `PorterDuff.Mode.SRC_IN` doesn't work with hardware acceleration.

Geometric Android Animations using the Canvas
Alexio Mota

2.8K 6

Romain Guy Follow
Dec 15, 2018 · 1 min read

This is incorrect, SRC_IN works with hardware acceleration. Your problem wasn't hardware acceleration, it's that the window surface is opaque. To achieve your effect the way you wrote it you need an intermediate buffer with an alpha channel. A hardware layer would therefore work and be much more efficient.

Source In

25 claps 1 response

LinearGradient positions

```
setContentView(View(this).also {
    it.background = PorterDuffTextDrawable(
        LinearGradient(
            0f, 0f, 0f, dp(64),
            intArrayOf(Color.GREEN, Color.YELLOW, Color.RED),
            null, // = floatArrayOf(0f, .5f, 1f)
            Shader.TileMode.CLAMP
        ),
        0×FF_3399FF.toInt(),
        "Превед!",
        PorterDuff.Mode.DARKEN
    )
    it.setLayerType(View.LAYER_TYPE_SOFTWARE, null)
}, FrameLayout.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT, dip(64)))
```

setLayerType(..., Paint)

```
setContentView(View(this).also {
    it.background = PorterDuffTextDrawable(
        LinearGradient(
            0f, 0f, 0f, dp(64),
            intArrayOf(Color.GREEN, Color.YELLOW, Color.RED),
            null,
            Shader.TileMode.CLAMP
        ),
        0×FF_3399FF.toInt(),
        "Превед!",
        PorterDuff.Mode.DARKEN
    )
    it.setLayerType(View.LAYER_TYPE_SOFTWARE, null)
}, FrameLayout.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT, dip(64)))
```

Indeterminate Progress

```
class ArcProgressDrawable(
    @ColorInt color: Int, @Px strokeWidth: Float
) : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG).also {
        it.color = color
        it.strokeWidth = strokeWidth
        it.style = Paint.Style.STROKE
    }









    }
}
```

Indeterminate Progress

```
class ArcProgressDrawable(
    @ColorInt color: Int, @Px strokeWidth: Float
) : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG).also {
        it.color = color
        it.strokeWidth = strokeWidth
        it.style = Paint.Style.STROKE
    }
    override fun draw(canvas: Canvas) {
        var (l, t, r, b) = bounds
        val pad = paint.strokeWidth / 2f

        canvas.drawArc(
            l + pad, t + pad, r - pad, b - pad,

        )

    }
}
```

Indeterminate Progress

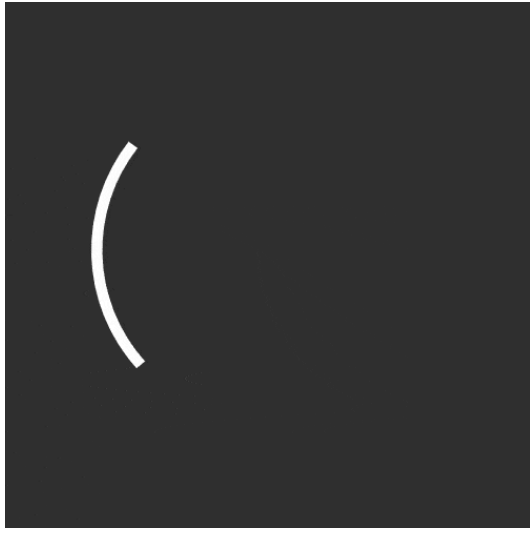
```
class ArcProgressDrawable(
    @ColorInt color: Int, @Px strokeWidth: Float
) : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG).also {
        it.color = color
        it.strokeWidth = strokeWidth
        it.style = Paint.Style.STROKE
    }
    override fun draw(canvas: Canvas) {
        var (l, t, r, b) = bounds
        val pad = paint.strokeWidth / 2f

        canvas.drawArc(
            l + pad, t + pad, r - pad, b - pad,
            ???, 80f,
        )

    }
}
```

Indeterminate Progress

```
class ArcProgressDrawable(
    @ColorInt color: Int, @Px strokeWidth: Float
) : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG).also {
        it.color = color
        it.strokeWidth = strokeWidth
        it.style = Paint.Style.STROKE
    }
    override fun draw(canvas: Canvas) {
        var (l, t, r, b) = bounds
        val pad = paint.strokeWidth / 2f

        canvas.drawArc(
            l + pad, t + pad, r - pad, b - pad,
            ???, 80f, false,
        )

    }
}
```

Indeterminate Progress

```
class ArcProgressDrawable(
    @ColorInt color: Int, @Px strokeWidth: Float
) : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG).also {
        it.color = color
        it.strokeWidth = strokeWidth
        it.style = Paint.Style.STROKE
    }
    override fun draw(canvas: Canvas) {
        var (l, t, r, b) = bounds
        val pad = paint.strokeWidth / 2f

        canvas.drawArc(
            l + pad, t + pad, r - pad, b - pad,
            ???, 80f, false, paint
        )

    }
}
```



Indeterminate Progress

```
class ArcProgressDrawable(
    @ColorInt color: Int, @Px strokeWidth: Float
) : Drawable() {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG).also {
        it.color = color
        it.strokeWidth = strokeWidth
        it.style = Paint.Style.STROKE
    }
    override fun draw(canvas: Canvas) {
        var (l, t, r, b) = bounds
        val pad = paint.strokeWidth / 2f
        val state = (uptimeMillis() % 1000).toInt() * 359 / 999
        canvas.drawArc(
            l + pad, t + pad, r - pad, b - pad,
            state.toFloat(), 80f, false, paint
        )
        invalidateSelf()
    }
}
```

Determinate Progress

```
class ArcProgressDrawable(
    @ColorInt color: Int, @Px strokeWidth: Float
) : Drawable() {
    @IntRange(from = 0L, to = 360L) var progress: Int = 0
        set(new) {
            if (new ≠ field) {
                require(new in 0..360) { "progress must ∈[0; 360], $new given" }
                field = new
                invalidateSelf()
            }
        }
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG).also { ... }
    override fun draw(canvas: Canvas) {
        var (l, t, r, b) = bounds
        val pad = paint.strokeWidth / 2f
        canvas.drawArc(
            l + pad, t + pad, r - pad, b - pad, -90f, progress.toFloat(), false, paint
        )
    }
    companion object {
        val PROGRESS = object : IntProperty<ArcProgressDrawable>("progress") {
            override fun get(drawable: ArcProgressDrawable): Int = drawable.progress
            override fun setValue(drawable: ArcProgressDrawable, value: Int) { drawable.progress = value }
        }
    }
}
```

Property animation

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val arcProgress = ArcProgressDrawable(Color.WHITE, dp(2))
    setContentView(
        View(this).also { it.background = arcProgress },
        FrameLayout.LayoutParams(MATCH_PARENT, MATCH_PARENT)
    )
    Handler(Looper.getMainLooper()).postDelayed({
        ObjectAnimator.ofInt(arcProgress, ArcProgressDrawable.PROGRESS, 0, 360)
            .also { it.interpolator = AccelerateDecelerateInterpolator() }
            .start()
    }, 5000)
```

Post-processing: dispatchDraw

```
override fun dispatchDraw(canvas: Canvas) {




    super.dispatchDraw(myCanvas)
    // TODO blur canvas
}
```

Post-processing: dispatchDraw

```
var myBitmap: Bitmap? = null
var myCanvas = Canvas()
override fun dispatchDraw(canvas: Canvas) {
    if (myBitmap.let { it == null || it.width < canvas.width || it.height < canvas.height }) {
        myBitmap?.recycle()
        myBitmap = Bitmap.createBitmap(canvas.width, canvas.height, Bitmap.Config.ARGB_8888)
        myCanvas.setBitmap(myBitmap)
    } else { myBitmap!!.eraseColor(Color.TRANSPARENT) }

    super.dispatchDraw(myCanvas)

    // TODO blur myBitmap

    canvas.drawBitmap(myBitmap, 0f, 0f, null)
}
```

Post-processing: invalidation

```
var myBitmap: Bitmap? = null
var myCanvas = Canvas()
override fun dispatchDraw(canvas: Canvas) {
    if (myBitmap.let { it == null || it.width < canvas.width || it.height < canvas.height }) {
        myBitmap?.recycle()
        myBitmap = Bitmap.createBitmap(canvas.width, canvas.height, Bitmap.Config.ARGB_8888)
        myCanvas.setBitmap(myBitmap)
    } else { myBitmap!!.eraseColor(Color.TRANSPARENT) }

    super.dispatchDraw(myCanvas)

    // TODO blur myBitmap

    canvas.drawBitmap(myBitmap, 0f, 0f, null)
}
override fun onDescendantInvalidated(child: View, target: View) {
  super.onDescendantInvalidated(child, target)
  invalidate() // SDK26+ reblur
}
override fun invalidateChildInParent(location: IntArray?, dirty: Rect?): ViewParent? {
  invalidate() // legacy reblur
  return super.invalidateChildInParent(location, dirty)
}
```

Screenshot

```
val v = ...
v.measure(
    View.MeasureSpec.UNSPECIFIED,
    View.MeasureSpec.UNSPECIFIED
)
v.layout(
    0, 0,
    v.measuredWidth, v.measuredHeight
)
val bitmap = Bitmap.createBitmap(
    v.width, v.height,
    Bitmap.Config.ARGB_8888
)
v.draw(Canvas(bitmap))
```

Directed by
SAINT P. ACADEMY

Executive Producer
Mike Gorünov


@Miha-x64


@miha_x64


@Harmonizr at android_ru, kotlin_lang, kotlin_lychee, spbpeerlab