

Elementi grafici

<https://developer.android.com/guide/topics/ui/declaring-layout>

<https://developer.android.com/guide/topics/ui/custom-components>

View

<https://developer.android.com/reference/android/view/View>

- A View occupies a rectangular area on the screen and is responsible for drawing and event handling.
 - The View class is a superclass for all GUI components in Android.
- Esempi
 - EditText
 - ImageView
 - TextView
 - Button
 - ImageButton
 - CheckBox

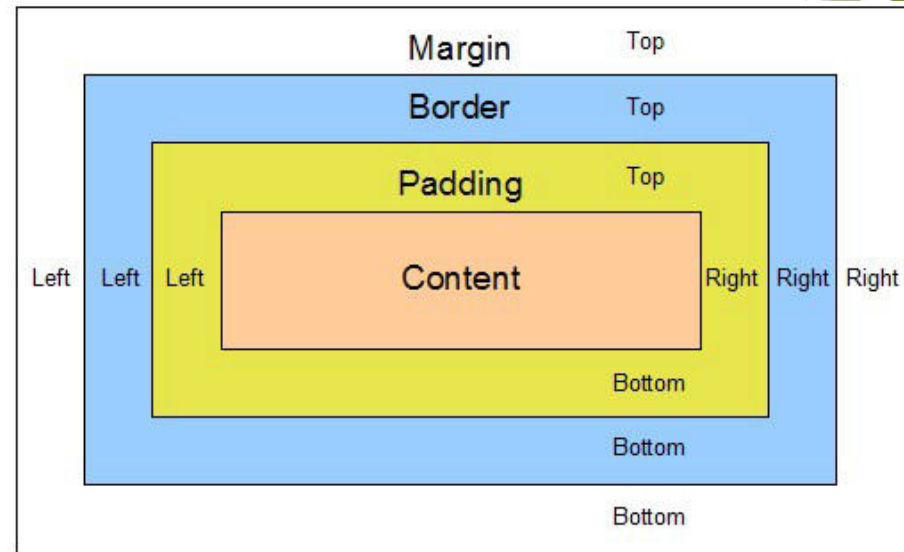
Operazioni comuni

- **Set properties:**
 - for example setting the text of a [TextView](#). The available properties and the methods that set them will vary among the different subclasses of views. Note that properties that are known at build time can be set in the XML layout files.
- **Set focus:**
 - The framework will handle moving focus in response to user input. To force focus to a specific view, call [requestFocus\(\)](#).
- **Set up listeners:**
 - Views allow clients to set listeners that will be notified when something interesting happens to the view. For example, all views will let you set a listener to be notified when the view gains or loses focus. For example, a Button exposes a listener to notify clients when the button is clicked.
- **Set visibility:**
 - You can hide or show views using [setVisibility\(int\)](#).

Attributi

- **id**
 - used to find specific views within the view tree.
 - Syntax: `android:id="@+id/my_id"`
- **height & width**
 - the height and width of the given view. These are necessary attributes for every view in a xml file.
 - Syntax: `android:layout_width="match_parent"`
`android:layout_height="match_parent"`
 - MATCH_PARENT means that the view wants to be as big as its parent (minus padding)
 - WRAP_CONTENT, which means that the view wants to be just big enough to enclose its content (plus padding).
 - A VALUE in "dp" i.e density independent pixels.

Padding & margin



- **Padding**

- the space inside the border, between the border and the actual view's content.

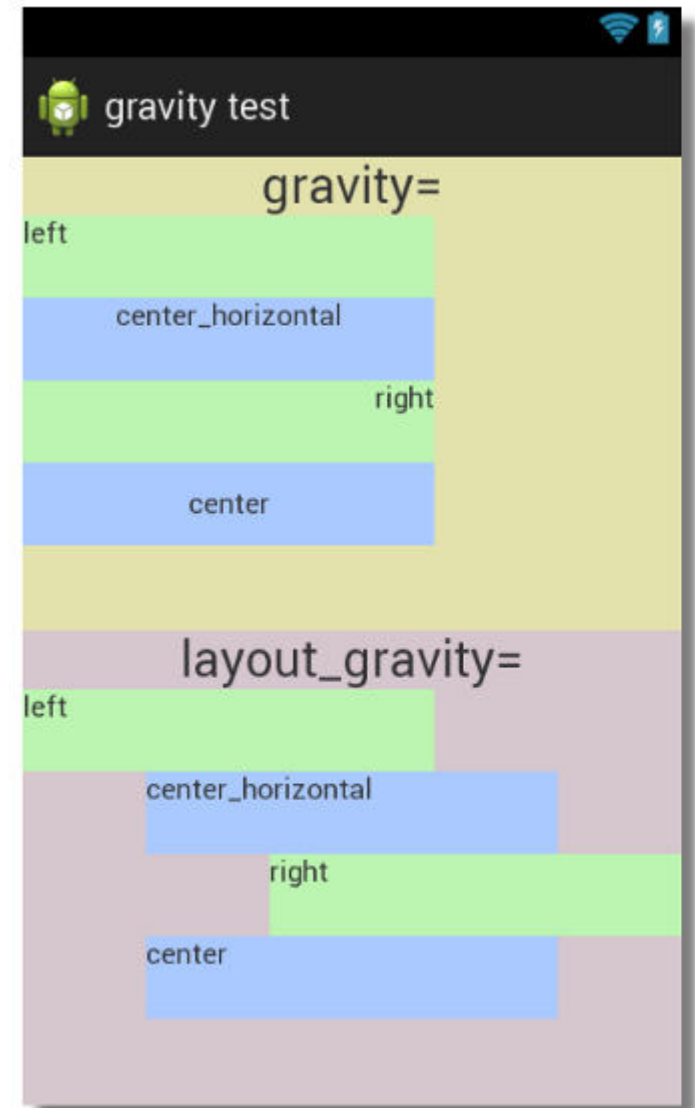
- **Margins**

- the spaces outside the border, between the border and the other elements next to this view.

Esempio: `android:padding="10dp"` `android:layout_margin="10dp"`

Gravity & Layout_gravity

- **android:gravity**
 - sets the gravity of the content of the View its used on.
- **android:layout_gravity**
 - sets the gravity of the View or Layout in its parent.



Some more attributes

- **android:maxHeight:**
 - An optional argument to supply a maximum height for this view.
- **android:maxLength**
 - An optional argument to supply a maximum width for this view.
- **android:scaleType**
 - Controls how the image should be resized or moved to match the size of this ImageView.
- **android:hint**
 - Hint text to display when the text is empty.
- **android:inputType**
 - The type of data being placed in a text field, used to help an input method decide how to let the user enter text.
- **android:textAppearance**
 - Base text color, typeface, size, and style.
- **android:textColor**
 - Text color.
- **android:textSize**
 - Size of the text.

ViewGroup

<https://developer.android.com/reference/android/view/ViewGroup.html>

- A ViewGroup is a special view that can contain other views (called children.)
 - The view group is the base class for layouts and views containers.

Linear Layout



Relative Layout 

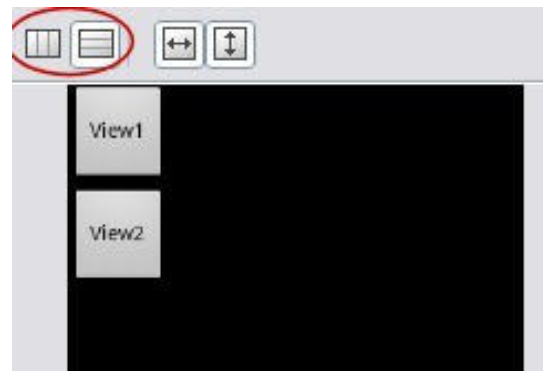


Web View



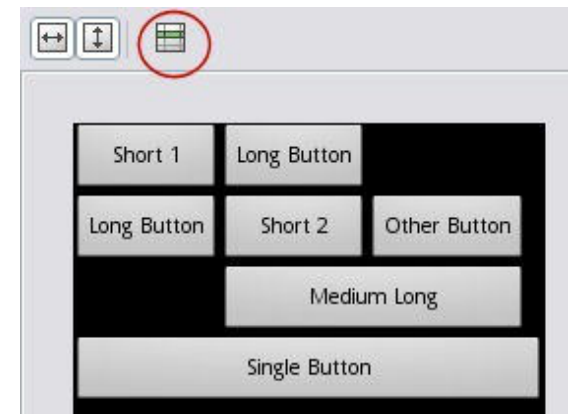
Layout

- Un layout è
 - una specializzazione di ViewGroup
 - posiziona le View contenute in un modo “specifico”



- Layout predefiniti

- LinearLayout
- RelativeLayout
- TableLayout
- FrameLayout
- ...



Container

- ScrollView
- AdapterView
 - ListView
 - RecyclerView
 - CardView
- ...

include

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/titlebar_bg"
    tools:showIn="@layout/activity_main" >

    <ImageView android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/gafricalogo" />

</FrameLayout>
```

layout/titlebar.xml

include

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/titlebar_bg"
    tools:showIn="@layout/activity_main" >

    <ImageView android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/gafricalogo" />

</FrameLayout>
```

layout/titlebar.xml

layout/main_activity.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/app_bg"
    android:gravity="center_horizontal">

    <include layout="@layout/titlebar" />

    <TextView android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        android:padding="10dp" />

    ...

</LinearLayout>
```



include

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/titlebar_bg"
    tools:showIn="@layout/activity_main" >

    <ImageView android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/gafricalogo" />

</FrameLayout>
```

layout/titlebar.xml

layout/main_activity.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/app_bg"
    android:gravity="center_horizontal">
```

override delle
proprietà!!!

```
<include android:id="@+id/news_title"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    layout="@layout/title" />
```

```
</LinearLayout>
```

merge

```
<merge xmlns:android="http://schemas.android.com/apk/res/android">

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/add" />

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/delete" />

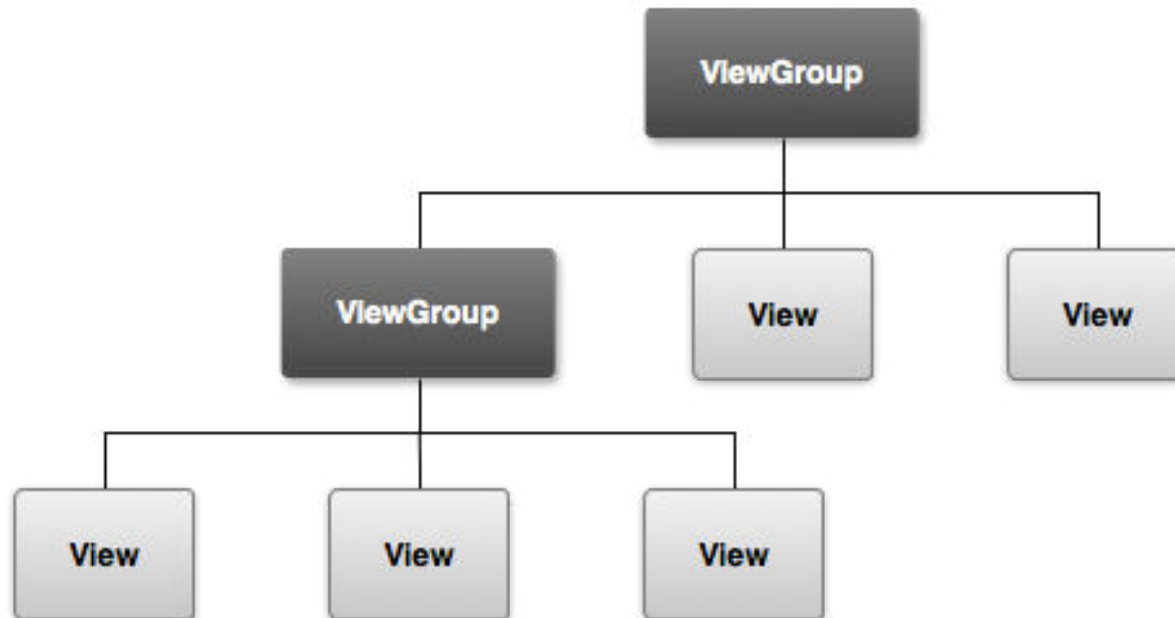
</merge>
```

- To avoid including a redundant view group
- Using the `<include/>` tag
 - the system ignores the `<merge>` element
 - places the two buttons directly in the layout

UI DRAWING

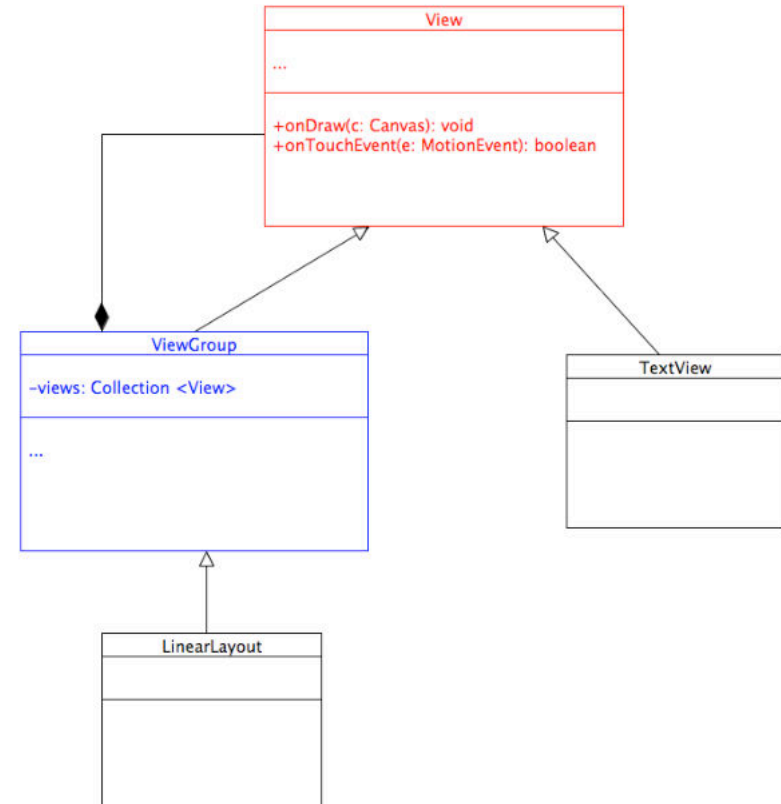
Albero delle view

- La grafica è composta da View e ViewGroup
 - Le view disegnano qualcosa sullo schermo e ci si può interagire
 - Le viewgroup sono contenitori invisibili per view e viewgroup
 - I componenti grafici formano un albero



Composite Pattern

- Le viewgroup estendono view
- Le viewgroup contengono una o più view
- Vantaggi
 - Realizzo collezioni di view
 - Non devo differenziare tra nodi e foglie
 - Posso chiamare un metodo su tutta la collezione



Attraversamento dell'albero

- Usando il composite pattern ad ogni Activity è associato l'albero delle view
 - Posso chiamare un metodo sulla radice ed a cascata viene chiamato su tutto l'albero
 - **draw!!**
 - L'attraversamento dell'albero cambia l'ordine di visualizzazione
- Android garantisce che
 - i nodi sono attraversati partendo dal root
 - i nodi più vicini al root sono attraversati prima
 - i nodi di pari livello sono attraversati secondo l'ordine di definizione

Processo di drawing

- Processo distribuito in tre passi
 - Determino le misure delle view sullo schermo
 - Determino la posizione delle view
 - Disegno le view
- Eseguo tutto in un singolo thread per non aver problemi di contesa
 - inibisco il controllo degli oggetti grafici dagli altri thread

Misura

- Il parent chiama **measure(int, int)**
 - Viene eseguito in metodo **onMeasure**
 - Può essere chiamato più volte
- Il parent comunica alle view se la loro misura
 - UNSPECIFIED – misura libera
 - EXACTLY – misura esatta
 - AT_MOST – misura massima
- **View.MeasureSpec**
 - `makeMeasureSpec` – metodo per fornire il tipo di misura
- Anche i layout devono calcolare la propria misura in base a quella delle view
 - `setMeasuredDimension`

Misure di una view

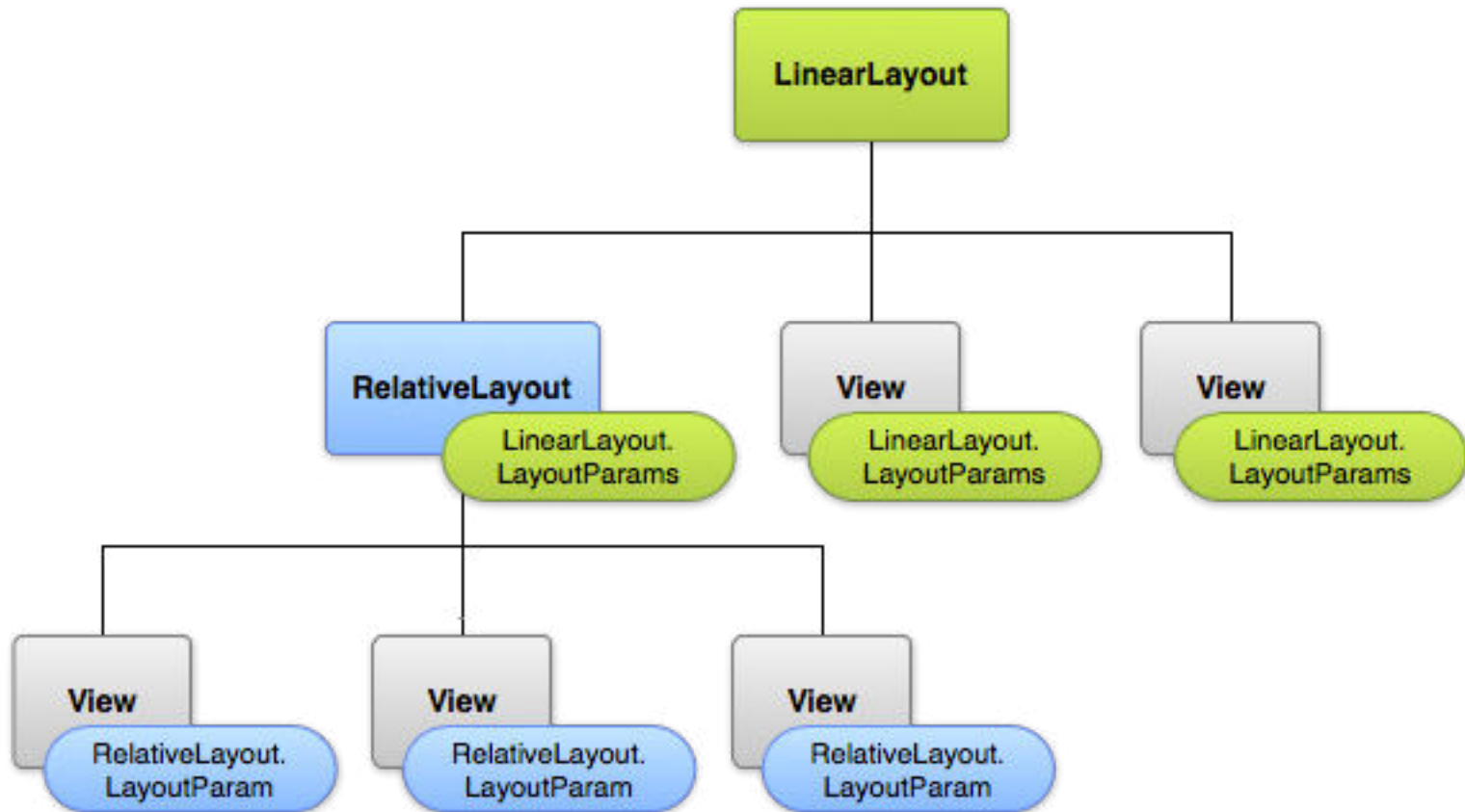
- Le view hanno due tipi di misure
 - **measured** quelle che vorrebbero avere
 - **effettive** quelle che realmente avranno
- Per ottenere le misure
 - `public final int getMeasuredWidth()`
 - `public final int getMeasuredHeight()`
 - `public final int getWidth()`
 - `public final int getHeight()`

Entrambe contengono il padding e non il margine

Posizionamento delle view

- Si attraversa l'albero una seconda volta per determinare la posizione delle view nota la loro dimensione
- Il parent chiama `layout(int, int, int, int)`
 - Viene eseguito in metodo `onLayout`
 - coordinate dei vertici top-left e bottom-right
- Le informazioni circa il layout sono memorizzate in istanze della classe `ViewGroup.LayoutParams`
 - Ogni layout definisce la sua con i parametri specifici

LayoutParam

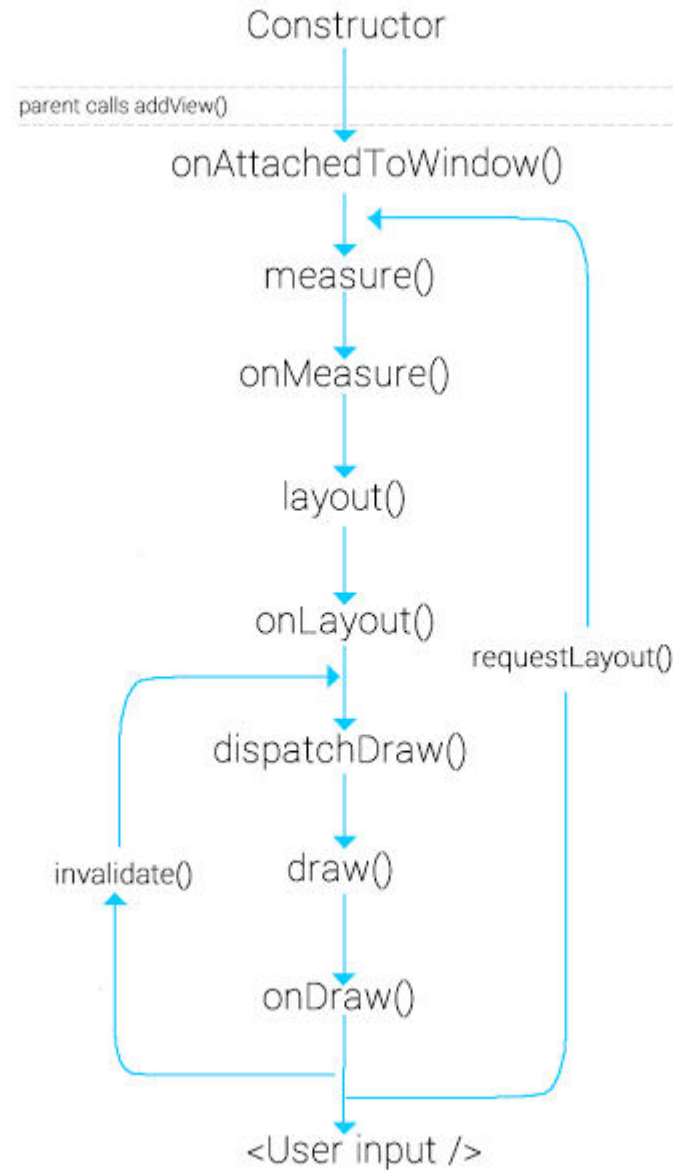


Disegno delle view

- Viene chiamato su tutto l'albero il metodo **draw**
 - questo chiama il metodo **onDraw**
- Tutte le view note le dimensioni e nota la posizione si disegnano

Ridisegnare le view

- **public void requestLayout ()**
 - richiede di ricominciare la fase di disegno partendo dal ricalcolo delle misure
- **public void invalidate ()**
 - serve a far ridisegnare le view
- **public void postInvalidate ()**
 - se sono in un thread differente da quello di grafica

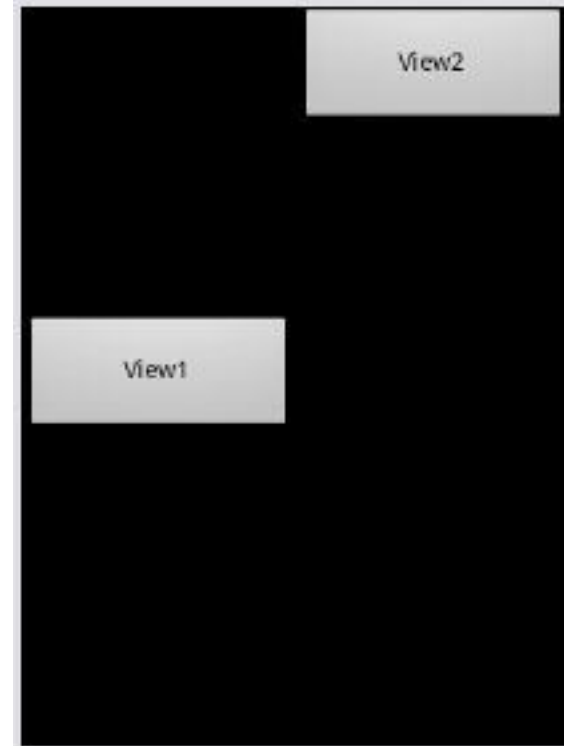


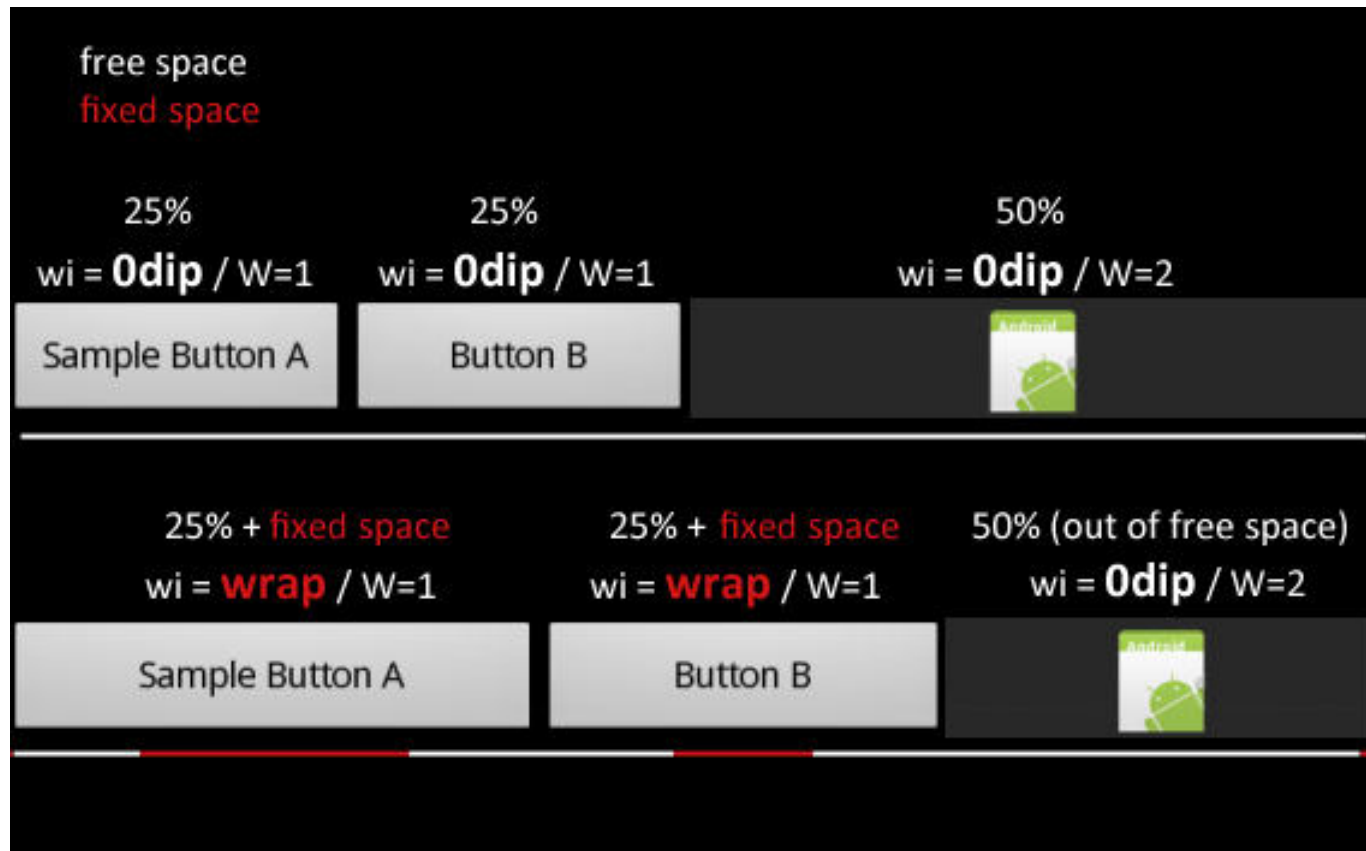
Esempio di Layout







- <https://developer.android.com/reference/android/view/ViewGroup.html>
 - Esempio di FrameLayout

Linear Layout

- Allinea le view contenute
- Si specifica il verso di allineamento
 - `android:layout_orientation`
 - `public void setOrientation (int orientation)`
- Si può specificare la dimensione relativa
 - `android:layout_weight`
- nell'altra direzione è possibile gestire l'allineamento
 - `android:layout_gravity`
- Progetto `LinearLayoutTest`





$w_i = \text{wrap} / W=1$ Sample Button A	$w_i = \text{wrap} / W=1$ Button B	$w_i = \text{wrap} / W=2$ 
$w_i = \text{wrap} / W=1$ Sample Button A	$w_i = \text{wrap} / W=1$ Button B	$w_i = \text{wrap} / W=1$ 
$w_i = \text{0dip} / W=1$ Sample Button A	$w_i = \text{0dip} / W=1$ Button B	$w_i = \text{wrap} / W=1$ 
$w_i = \text{wrap}$ Sample Button A	$w_i = \text{wrap}$ Button B	$w_i = \text{any} / W=1$ 
$w_i = \text{wrap}$ Sample Button A	$w_i = \text{wrap}$ Button B	$w_i = \text{match_parent/fill_parent}$ 
$w_i = \text{wrap} / W = 2$ Sample Button A	$w_i = \text{wrap} / W = 2$ Button B	$w_i = \text{0dip} / W = 1$ 
$w_i = \text{0dip} / W = 2$ Sample Button A	$w_i = \text{0dip} / W = 2$ Button B	$w_i = \text{0dip} / W = 1$ 



wrap_content for all views



wrap_content for all views



wrap_content, 0dp, 0dp



wrap_content, 0dp, 0dp



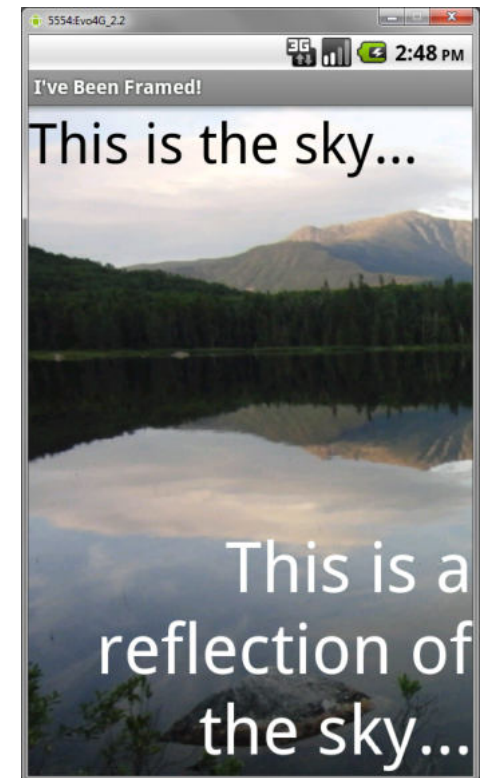
wrap_content for all views

Relative Layout

- Posizionamento delle view relativo
 - ad altre view
 - al parent

Frame layout

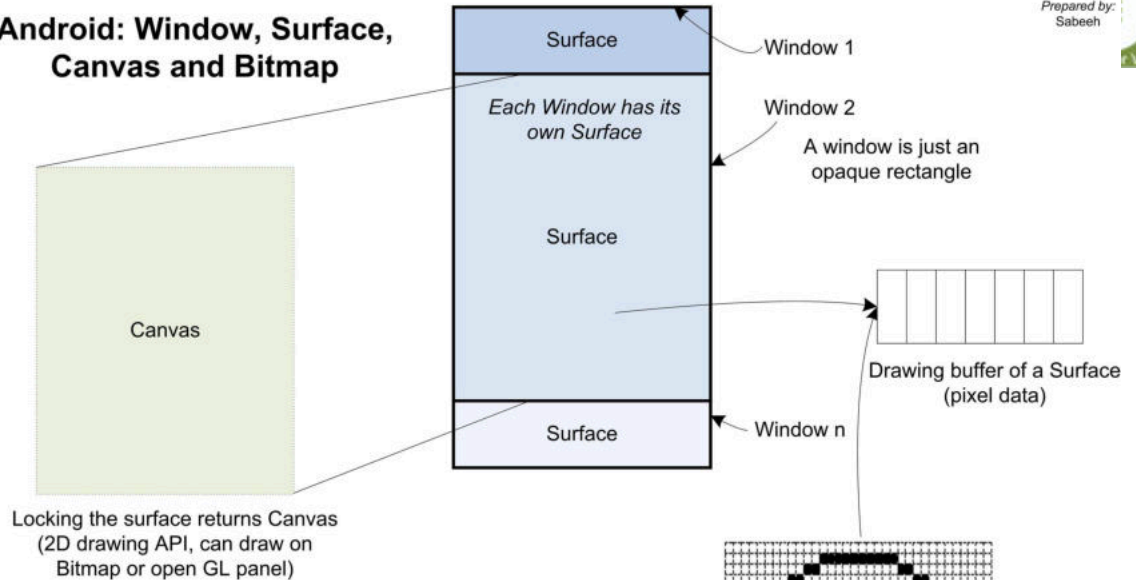
- Singola finestra
 - tutte le view sono poste una sull'altra
 - posso solo controllare la loro gravity



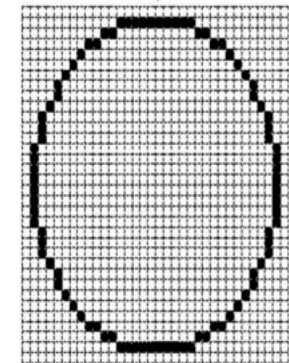
TOPIC AVANZATI

Window

Android: Window, Surface, Canvas and Bitmap



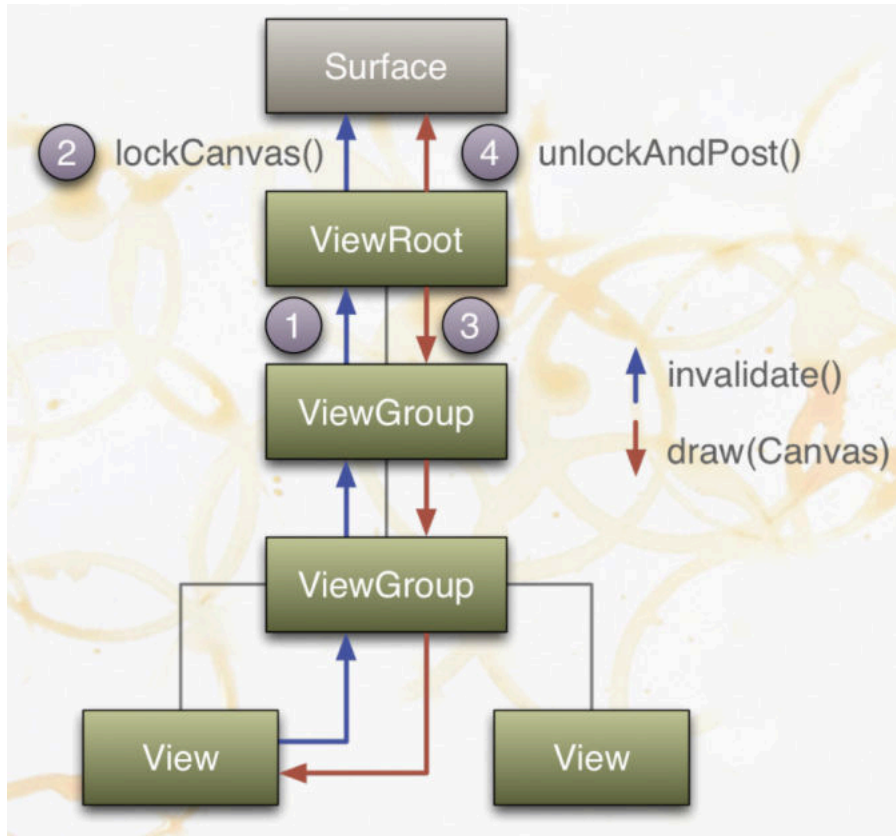
- The Surface is locked, which returns a Canvas that can be used to draw into it.
- All the logic on how to draw a circle, or a box, etc is contained inside Canvas.
- Canvas can't write to a surface.
- A canvas draws on a Bitmap or an open GL container.
- So, a Bitmap is created and pointed to the current drawing buffer of the Surface. Now the returned canvas will draw into this Bitmap.
- A window has a single view hierarchy attached to it.
- A draw traversal is done down the view hierarchy, handing this Canvas down for each view to draw its part of the UI.
- Once done, the Surface is unlocked and posted.
- Surface Flinger now composite the screen using the buffer just drawn.



Bitmap (Raster)
A Bitmap can point to its own pixel data or as in this case to the pixel data it does not own. Canvas returned by surface can draw into this Bitmap.

<https://stackoverflow.com/questions/4576909/understanding-canvas-and-surface-concepts#answers>

Surface



<https://www.youtube.com/watch?v=duefsFTJXzc>