**ANIMAZIONI**

**Come funziona il meccanismo di layout e la relativa fase di misurazione delle view?**

Il meccanismo di layout si compone di 3 fasi:

**Cosa è l’albero delle view? Quali sono e come funzionano le due fasi necessarie per la visualizzazione del layout?**

L’albero delle view è l’insieme dei widget che formano un determinato layout, strutturato secondo una gerarchia ad albero in quanto ogni file XML ne conserva la struttura. Alla radice dell’albero si trova un singolo ViewGroup, al livello successivo sono posti i suoi figli, al livello successivo sono posti i suoi “nipoti” e così via, fin quando non si raggiungono le foglie dell’albero.

Le fasi necessarie per la visualizzazione del layout sono tre:

1. ***Misurazione***. fase in cui viene fatta una visita top-down dell’albero, per cui si misura width ed height della radice, e si esegue la stessa misurazione per ogni sottoalbero contenuto nell’albero delle view, ricorsivamente. In questa fase si effettuano due misurazioni: una misurazione “temporanea” che indica quanto grande, ogni view, vorrebbe essere, e una misurazione che indica quanto grande ogni view sarà realmente;
2. ***Posizionamento***: fase in cui viene fatta una visita top-down dell’albero, per cui si posiziona dapprima la radice, ed è proprio la radice a posizionare ognuno dei suoi figli, così come ogni figlio posizionerà ognuno dei suoi figli, e così via fin quando non si raggiungono le foglie dell’albero;
3. ***Disegno***. fase in cui si disegna ogni view nel layout dell’interfaccia.

**Si spieghi come avviene la misurazione e il posizionamento delle view di un layout. Perché in alcuni casi i metodi v.getWidth e v.getHeight, dove v è una view del layout, usati in onCreate() restituiscono 0?**

Le view di un layout, prima che vengano effettivamente disegnate nell’interfaccia utente attraverso la fase di draw, devono essere prima misurate, poi posizionate correttamente nell’UI. Bisogna prima specificare che ogni layout possiede il cosiddetto albero delle view (siccome ogni file XML è strutturato gerarchicamente in un albero).

La fase di ***misurazione*** ricava la misura che le varie view richiedono e calcola la misura reale che le varie view avranno, in relazione alla view genitore e ad i suoi fratelli (cioè, le view che risiedono nello stesso livello dell’albero): si parte dalla radice e si analizza ognuno dei sottoalberi di ogni figlio ricorsivamente, fin quando tutte le view sono state misurate.

La fase di ***posizionamento*** prevede il posizionamento delle view del layout secondo le misure calcolate nella fase precedente.

In alcuni casi, nel metodo onCreate() non è ancora possibile ricavare le misure (width ed height) delle varie view, in quanto è possibile che esse non siano state ancora posizionate e disegnate nel layout, anche dopo aver invocato il metodo setContentView().

**Che cosa è e a cosa serve l’oggetto Canvas? Si illustri, con sufficienti dettagli, un esempio in cui è necessario utilizzare un oggetto Canvas.**

L’oggetto ***Canvas*** consente di disegnare una view customizzata in un determinato punto del layout.

Lo step più impostante per disegnare una view personalizzata è eseguire l’override del metodo ***onDraw()***, che riceve come parametro un oggetto ***Canvas*** che la view può usare per disegnare sé stessa. La classe ***Canvas*** definisce metodi per disegnare testo, linee, bitmap e molte altre grafiche primitive. Si possono usare i suoi metodi in ***onDraw()*** per creare la propria UI customizzata.

Prima di poter chiamare qualsiasi metodo di disegno, tuttavia, è necessario creare uno o più oggetti ***Paint***.

Ad esempio, un ***Canvas*** fornisce un metodo per disegnare una linea, mentre ***Paint*** fornisce metodi per definire il colore della linea; ***Canvas*** fornisce un metodo per disegnare un rettangolo, mentre ***Paint*** definisce se riempire tale rettangolo con un colore o lasciarlo vuoto. Semplicemente, ***Canvas*** definisce le forme che si possono disegnare sullo schermo, mentre ***Paint*** definisce colore, stile, font e così via, per ogni forma si vuole disegnare.

Durante il corso è stato analizzato un esempio in cui si disegnava un Pentagramma su cui era possibile aggiungere varie note.

**Si spieghi come si creano le animazioni grafiche (magari descrivendo tutte le possibilità) in Android possibilmente facendo un esempio concreto (magari usando solo una o due delle varie possibilità descritte).**

Android permette di definire delle animazioni da applicare alle immagini, descritte con file XML:

* Rotazione
* Traslazione
* Scaling
* Trasparenza

Con controlli di vario tipo usufruendo dei parametri, come pivot, velocità, etc….

**Come funzionano le animazioni? Come si può animare un oggetto grafico in Android? Si faccia un esempio scegliendo una specifica animazione.**

Le animazioni consentono ad una determinata view di aggiungervi degli effetti speciali del tipo, tra gli altri: rotazione, traslazione, scaling, trasparenza. Per creare un’animazione, occorre creare un file di layout XML: alla radice dell’albero dev’essere posto un elemento <set> (oppure <objectAnimator> e <valueAnimator>), che indica l’insieme di animazioni da eseguire, al cui interno è possibile porre <animator> ed <objectAnimator>, ossia le singole animazioni, o ulteriori elementi <set>.

Ogni file d’animazione può essere memorizzato in un oggetto Animation. Per assegnare un’animazione ad un oggetto animation, occorre invocare il metodo statico AnimationUtils.loadAnimation(Context context, int resource), che ritorna un oggetto Animation.

Un’animazione può essere utilizzata sulla maggior parte delle View di un layout, utilizzando il metodo View.startAnimation(Animation animation).

|  |
| --- |
| *<rotate*  *android:startOffset="0"*  *android:duration="4000"*  *android:fromDegrees="0"*  *android:toDegrees="180"*  *android:pivotX="50%"*  *android:pivotY="50%"*  */>* |

Nella nostra MainActivity verrà istanziato un oggetto Animation che verrà eguagliato nel seguente modo:

|  |
| --- |
| *AnimationUtils.loadAnimation(getApplicationContext(), R.anim.rotazione);* |

Verrà definito un metodo invocato al click del bottone che starterà l’oggetto con la relativa animazione.

**Si scriva un file XML con un’animazione che permette di traslare prima un oggetto prima in orizzontale per 500 pixels in 1 secondo e poi in verticale per 300 pixel in 2 secondi. Normalmente dopo l’animazione l’oggetto ritorna automaticamente nella posizione iniziale; cosa si può fare per farle rimanere nella posizione finale dell’animazione?**

|  |
| --- |
| *<set xmlns:android="http://schemas.android.com/apk/res/android"*  *android:shareInterpolator="false"*  *<translate*  *android:startOffset="0"*  *android:duration="1000"*  *android:fromXDelta="0"*  *android:fromYDelta="0"*  *android:toXDelta="500"*  *android:toYDelta="0" />*  *<translate*  *android:startOffset="1000"*  *android:duration="2000"*  *android:fromXDelta="0"*  *android:fromYDelta="0"*  *android:toXDelta="0"*  *android:toYDelta="300" />*  *</set>* |

Per far sì che un oggetto traslato non ritorni nella sua poizione iniziale dopo l’animazione, è possibile procedere come segue:

1. Aggiungere un ***AnimationListener*** all’animazione che effettua la traslazione:

|  |
| --- |
| *animation.setAnimationListener(new Animation.AnimationListener(){*  *@Override*  *public void onAnimationStart(Animation animation){ }*  *@Override*  *public void onAnimationEnd(Animation animation){ }*  *@Override*  *public void onAnimationRepeat(Animation animation){ }*  *});* |

1. Nel metodo ***onAnimationEnd(),*** chiamare il metodo public void layout(int left, int top, int right, int bottom) sulla view traslata, ed aggiornare i parametri opportuni. Un altro modo è settare i parametri di layout ricavando i ***LayoutParams***, settandoli ed aggiungerli alla View.

**Il seguente snippet di codice esegue prima un’animazione e poi rimuove l’oggetto dalla view parent (gli oggetti image, animation e parentView sono stati in precedenza opportunamente inizializzati):**

***…***

***image.startAnimation(animation);***

***parentView.removeViewAt(0);***

***….***

**Tuttavia l’effetto è quello di rimuovere immediatamente l’immagine senza dare il tempo all’animazione di essere eseguita. Perché accade ciò? Come si può ovviare al problema?**

Il problema precedentemente descritto accade in quanto il metodo ***image.startAnimation()*** non interrompe l’esecuzione del codice, e la duration dell’animazione viene sorvolata dall’esecuzione del programma. Per ovviare al problema, si potrebbe aggiungere un ***AnimationListener*** all’animazione, ed al termine (eseguendo l’override del metodo listener ***onAnimationEnd())*** rimuovere l’oggetto dalla view ***parent***. Un altro modo potrebbe essere l’implementazione di un ***AsyncTask<ImageView, Integer, Boolean>,*** in modo tale che in ***doInBackground(ImageView imageView)*** venga avviata l’animazione e ritornato true, e in ***onPostExecute(Boolean boolean)*** venga rimosso l’oggetto se boolean == true.

**La seguente classe implementa una view customizzata per un pentagramma. Così come mostrato l’oggetto Pentagramma viene disegnato con una nota in una specifica posizione (alle coordinate 350,78). Descrive le modifiche da fare per fare in modo che inizialmente il pentagramma sia vuota e che successivamente si possano aggiungere delle note al pentagramma.**

**public class Pentagramma extends View {**

**int MARGIN\_TOP= 50;**

**int MARGIN\_H= 40;**

**int LINE\_SPACING = 8;**

**int NOTEHEAD\_SIZE = 6;**

**int STEM\_LENGTH = 30;**

**public class Nota { … }**

**ArrayList<Nota> notesList = new ArrayList<Nota>();**

**public Pentagramma(Context c, int screen\_w, int screen\_h) {**

**super(c);**

**setMinimumWidth(screen\_w);**

**setMinimumHeight(150);**

**Nota nota = new Nota(350, 78);**

**notesList.add(nota);**

**}**

**@Override**

**protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) { ... }**

**@Override**

**protected void onLayout(boolean b, int x1, int y1, int x2, int y2) { … }**

**@Override**

**protected void onDraw(Canvas canvas) {**

**Paint paint = new Paint();**

**paint.setColor(Color.BLACK);**

**int h = MARGIN\_TOP;**

**int x\_start = MARGIN\_H;**

**int x\_end = canvas.getWidth() - MARGIN\_H;**

**for (int i = 0; i<5; i++ ) {**

**canvas.drawLine(x\_start,h,x\_end,h,paint);**

**h = h + LINE\_SPACING;**

**}**

**Nota n = notesList.get(0);**

**int note\_position\_horizzontal = n.getH();**

**int note\_position\_vertical = n.getV();**

**canvas.drawCircle(note\_position\_horizzontal, note\_position\_vertical, NOTEHEAD\_SIZE, paint);**

**canvas.drawLine(note\_position\_horizzontal+5, note\_position\_vertical-STEM\_LENGTH,**

**note\_position\_horizzontal+5, note\_position\_vertical, paint);**

**}**

Prima di tutto, occorre eliminare le ultime cinque righe del codice, ed eliminare le ultime due righe del costruttore del Pentagramma

Per inserire note in un pentagramma vuoto, è necessario implementare o un onclick listener o un ***ontouch listener*** sulla view del pentagramma: è preferibile utilizzare il secondo, in quanto il ***MotionEvent*** consentirà di ricavare il valore di asse x e y del punto in cui è stato effettuato il tocco.

Dunque, Pentagramma implementerà l’interfaccia del ***listener ontouch (OnTouchListener),*** per qui definirà il metodo ***onTouch***: al suo interno ricaviamo X ed Y del punto in cui è stato effettuato il tocco, istanziamo una nuova nota passando i due valori precedenti, aggiungiamo la nota all’ArrayList<Nota> notesList e terminiamo con un invalidate() del pentagramma: questo fa sì che il pentagramma venga ridisegnato, per cui all’interno di ***onDraw(Canvas canvas)*** inseriamo un ***for (int i=0; i<notes.size(); i++)*** al cui interno si disegnano tutte le note inserite fino a quel momento (potrebbe risultare inefficiente: si potrebbe, per questo motivo, effettuare il disegno della singola nota effettuando gli accorgimenti opportuni).

**Si fornisca un file XML per un’animazione che prima ruota la view di 180°e poi la sposta orizzontalmente di 200dp.**

|  |
| --- |
| *<set xmlns:android="http://schemas.android.com/apk/res/android"*  *android:shareInterpolator="false" >*  *<rotate*  *android:startOffset="0"*  *android:duration="xxxx"*  *android:fromDegrees="0"*  *android:toDegrees="180"*  *… />*  *<translate*  *android:startOffset="xxxx"*  *android:duration="yyyy"*  *android:fromXDelta="0"*  *android:fromYDelta="0"*  *android:toXDelta="200"*  *android:toYDelta="0" />*  *</set>* |

**Si scriva un file xml per la seguente animazione di un oggetto drawable:**

1. **rotazione di 2 giri completi a destra, dal tempo 0 al tempo 2 sec**
2. **traslazione di 300px a destra, dal tempo 3 sec al tempo 4 sec,**
3. **rotazione di 2 giri completi a sinistra, dal tempo 3 sec al tempo 5 sec.**

|  |
| --- |
| *<rotate*  *android:startOffset="0"*  *android:duration="2000"*  *android:fromDegrees="0"*  *android:toDegrees="720"*  */>*  *<translate*  *android:startOffset="3000"*  *android:duration="1000"*  *android:fromXDelta="0"*  *android:fromYDelta="0"*  *android:toXDelta="300"*  *android:toYDelta="0"*  */>*  *<rotate*  *android:startOffset="3000"*  *android:duration="2000"*  *android:fromDegrees="0" (720)*  *android:toDegrees="-720" (0)*  */>* |

**Si scriva un file animation.xml che permette di implementare la seguente animazione di una view: spostamento orizzontale di 300 px verso destra, seguito da una rotazione di 2 giri completi in senso orario con pivot al centro dell’oggetto, a sua volta seguita da un nuovo spostamento in orizzontale di 300px verso sinistra che riporta l’oggetto nella posizione originaria.**

|  |  |
| --- | --- |
| *<translate*  *android:startOffset="0"*  *android:duration="2000"*  *android:fromXDelta="0"*  *android:fromYDelta="0"*  *android:toXDelta="300"*  *android:toYDelta="0"*  */>*  *<rotate*  *android:startOffset="2000"*  *android:duration="5000"*  *android:fromDegrees="0"*  *android:toDegrees="720"*  *android:pivotX= "80%"*  *android:pivotY= "50%"*  */>* | *<translate*  *android:startOffset="7000"*  *android:duration="2000"*  *android:fromXDelta="0"*  *android:fromYDelta="0"*  *android:toXDelta="-300"*  *android:toYDelta="0"*  */>* |

**Si abbozzi (nel senso che si possono tralasciare i dettagli implementativi scrivendo cose del tipo “disegna una linea orizzontale ad un terzo dell’altezza”, oppure “disegna una X nella casella i,j”) la classe TrisView che estende una View e che fornisce dei metodi, addX(i,j) e addO(i,j), per visualizzare una “X” o una “O” in ognuno dei 9 riquadri. Un esempio di utilizzo della classe è riportato di seguito:**

**public class MainActivity extends Activity {**

**@Override**

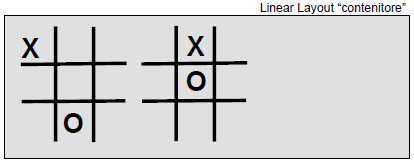
**protected void onCreate(Bundle savedInstanceState) {**

**super.onCreate(savedInstanceState);**

**setContentView(R.layout.activity\_main);**

**LinearLayout contenitore = (LinearLayout) findViewById(R.id.contenitore);**

**TrisView tris1 = new TrisView(300,300);**

**TrisView tris2 = new TrisView(300,300);**

**contenitore.addView(tris1);**

**contenitore.addView(tris2);**

**tris1.addX(1,1);**

**tris1.addO(3,2);**

**tris1.invalidate();**

**tris2.addX(1,2);**

**tris2.addO(2,2);**

**tris2.invalidate();**

**}**

**}**

**public class TrisView extends View {**