

<RelativeLayout

android:layout_width="match_parent"
android:layout_height="match_parent">

<LinearLayout

android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:orientation="horizontal"
android:id="@+id/linear"
android:layout_centerInParent="true">

<Button

android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Pulsante 1"
android:onClick="azione1"/>

<Button

android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Pulsante 2"
android:onClick="azione2"/>

</LinearLayout>

<TextView

android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Square"

~~android:layout_below="@id/linear"~~

android:layout_above="@id/linear"

android:layout_alignHorizontal="true" />

<TextView

android:layout_width="match_parent"

android:layout_height="match_parent"

android:text="Sito"

android:layout_below="@id/linear"

android:layout_alignHorizontal="true" />

</RelativeLayout>

~~Il primo~~ L'informazione principale che ci viene data in input nel metodo `OnClickListItem` non è il parametro `position`. Grazie ad esso conosciamo la posizione delle view all'interno delle `ListView` e possiamo utilizzarlo per ottenere l'oggetto che è stato cliccato dalle strutture dati collegate all'adapter.

Ciò però non basta se si usano dei listener per i singoli elementi, dato che esso manda in input solo le `View`. Per arrivare a questo problema possiamo, nel ~~metodo~~ metodo `getView` della classe `CustomAdapter`, settare come tag ~~la posizione~~ di ogni singolo elemento la posizione. In questo modo, nei metodi che rispondono al click del singolo elemento, grazie al metodo `v.getTag()`, possiamo ottenere la `position`.


```
public class AlteraActivity extends Activity {
```

```
    @Override
```

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

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.altera_layout);
```

```
        Intent i = getIntent();
```

```
        tvStampa = (TextView) findViewById(R.id.tvStampa);
```

```
        tvNumero = (TextView) findViewById(R.id.tvNumero);
```

```
        int numero = i.getIntExtra("Intero", 0);
```

```
        String str = i.getStringExtra("Stampa");
```

```
        tvStampa.setText(str);
```

```
tvStampa
```

```
        tvNumero.setText(numero);
```

```
    }
```

```
    TextView tvStampa, tvNumero;
```

```
}
```

```
public class MainActivity extends Activity {
```

```
    @Override
```

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

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
```

```
        FragmentManager fm = getSupportFragmentManager();
```

```
        FrameLayout c1 = (FrameLayout) findViewById(R.id.container1);
```

```
        FrameLayout c2 = (FrameLayout) findViewById(R.id.container2);
```

```
        TextView t1 = new TextView();
```

```
        TextView t2 = new TextView();
```

```
        t1.setText("t1");
```

```
        t2.setText("t2");
```

```
        FragmentTransaction ft = fm.beginTransaction();
```

```
        ft.add(c1, t1);
```

```
        ft.add(c2, t2);
```

```
        ft.commit();
```

```
    }
```

```
}
```


Antonio, CIRCU

QUESTO 6

Il metodo `invalidate()` viene chiamato quando viene applicata una modifica alla view come ad esempio la posizione. Ad esempio, creiamo una applicazione dove al trascinare del dito sullo schermo viene spostata una view. Ora, per poter fare ciò, ogni qualvolta si registra uno spostamento del dito sullo schermo, chiamiamo il metodo `invalidate()`, che a sua volta chiamerà il metodo `onDraw()` della view e il metodo `requestLayout` di tutte le view padre.

La classe `AsyncTask` nasce per semplificare le comunicazioni tra thread secondari e thread main. Questo perché i thread secondari hanno le necessità di svolgere operazioni "lente", ma non possono in nessun modo modificare l'UI, infatti, l'UI è possibile modificarla solo dal thread main. Questo perché il thread secondario però potrebbe avere le necessità di modificare l'UI per poter notificare l'utente del suo stato di avanzamento. Per fare ciò, quindi, il thread secondario deve comunicare il suo stato al thread main che potrà e modificare l'UI in base ad esso.

La classe `AsyncTask` serve appunto a semplificare queste comunicazioni fornendo alcuni metodi che saranno chiamati dal thread main e altri dal thread secondario. Questi sono:

- `onPreExecute`, chiamato dal thread main prima dell'esecuzione del thread secondario. Può essere usato per rendere visibile una `ProgressBar` ad esempio;
- `doInBackground`, chiamato dal thread secondario. Qui vengono effettuate le operazioni e lavori del thread secondario. Esso potrà comunicare il suo stato usando il metodo `publishProgress` passando come argomento in input il suo stato. Questo metodo fa sì che il thread main esegua il metodo `onProgressUpdate`;
- `onProgressUpdate`, chiamato dal thread main. Qui il thread main riceve lo stato di avanzamento del thread secondario, e quindi modifica l'UI in base ad esso. Ad esempio modifica lo stato di una `ProgressBar`.
- `onPostExecute`, chiamato dal thread main dopo la fine dell'esecuzione di `doInBackground`. Qui, ad esempio, il thread main, rende invisibile la `ProgressBar`.

Vi sono due modi per creare un'animazione in android.

Il primo metodo è ~~chiamato~~ l'utilizzo di un file XML. Esso deve essere situato nella directory res/anim. La struttura del file è costituita da un nodo <set> che include in se tutte le animazioni da applicare alla view. Esse vengono descritte in modi come <rotation>, <translate>, <alpha>, <~~scale~~ scale> etc... Ognuna di esse ha degli attributi ben precisi che ~~devono~~ servono appunto per impostare il tipo di animazione. Alcuni attributi in comune sono tipo, startOffset, che indica in ms il delay di start dell'animazione, e duration, che indica appunto ~~il~~ la durata dell'animazione.

Il secondo metodo è in modo programmatico, ovvero attraverso il codice JAVA.

Per settare l'animazione di un widget, ci basterà usare il metodo animate() sul widget che intendiamo animare. Questo metodo ci restituisce un riferimento all'oggetto ViewAnimationProperty, il quale offre metodi per animare le nostre view come .rotate(), .alpha() etc...

CIRILLO, ANTONIO

QUESITO 9

Le classe `Purse` le possiamo vedere come un puntatore ad un set di risultati forniti da una query (simile alla classe `ResultSet`).

È quindi molto utile se si ha la necessità di puntare (spostarsi) tra i risultati di una query grazie ai suoi metodi come `moveToFirst`. Grazie ad esso possiamo iterare i risultati e poterli quindi gestire.

I Service possiamo definirli come l'opposto dell'activity. Sono delle operazioni che vengono eseguite totalmente in background, e quindi non necessitano dell'interazione con l'utente. Esistono due tipi di service: started e bounded:

- started, ovvero un service che l'activity deve monitorare dall'inizio della sua esecuzione alle fine;
- bounded per essi vengono attivati solo quando un'app esterne vuole interagire con loro.