Linee guida per superare lo scritto di Android.

1. Come definire il layout di un'applicazione.

Il layout di un'applicazione può essere definito sia staticamente tramite file XML (metodo dichiarativo) sia nel codice sorgente, ovvero nelle classi Java (metodo programmatico).
Con un file XML vengono costruiti layout statici. Il vantaggio di questo metodo è la netta separazione tra il codice dell'applicazione e il suo layout. Questo metodo ha però come svantaggio il fatto che non è possibile creare un layout dinamico.
In modo programmatico, ossia tramite istruzione nel programma che saranno eseguite a runtime, è possibile costruire layout dinamici. Il grande svantaggio di questo metodo è che non vi è una separazione tra il codice dell'applicazione e il suo layout, ciò comporta una gestione più complicata rispetto al metodo dichiarativo.

2. Quali sono i Layout più comuni.

- **LinearLayout:** contiene un insieme di elementi che distribuisce in maniera sequenziale dall'alto verso il basso o da sinistra a destra. È un layout molto semplice e piuttosto naturale per i display di smartphone e tablet.
- **GridLayout:** altro layout piuttosto semplice. Inquadra gli elementi in una tabella e quindi è particolarmente adatto a mostrare strutture regolari suddivise in righe e colonne.
- **RelativeLayout:** sicuramente il più flessibile e moderno. Adatto a disporre in maniera meno strutturata gli elementi. Gli elementi al suo interno, infatti, vengono posizionati in relazione l'uno all'altro o rispetto al contenitore, permettendo un layout fluido che si adatta bene a display diversi.
- **ConstraintLayout:** simile al RelativeLayout. Permette di specificare la posizione dei suoi elementi attraverso dei vincoli. Utile per evitare una gerarchia di layout innestati troppo profonda. Molto comoda quando si lavora con l'editor grafico.

3. Come creare il layout che ti viene richiesto.

Utilizzare il RelativeLayout o al massimo il GridLayout in alcuni casi. Il RelativeLayout è molto utile per poter posizionare i vari widget in posizioni ben precise nel layout. Ecco una lista di tutti gli attributi utili per rappresentare i vari layout richiesti:

3.1. android:layout_alignParentBottom: se viene settato a true, fa sì che il bordo inferiore di questa view corrisponda al bordo inferiore del genitore.

3.2. android:layout_alignParentEnd: se viene settato a true, fa sì che il bordo finale di questa view corrisponde al bordo finale del genitore.

3.3. android:layout_alignParentStart: se viene settato a true, fa sì che il bordo iniziale di questa view corrisponde al bordo iniziale del genitore.

3.4. android:layout_centerInParent: se viene settato a ture, fa sì che la view viene posizionata al centro del genitore.

3.5. android:layout_above: il widget viene posizionato sopra al widget di cui verrà inserito l'id.

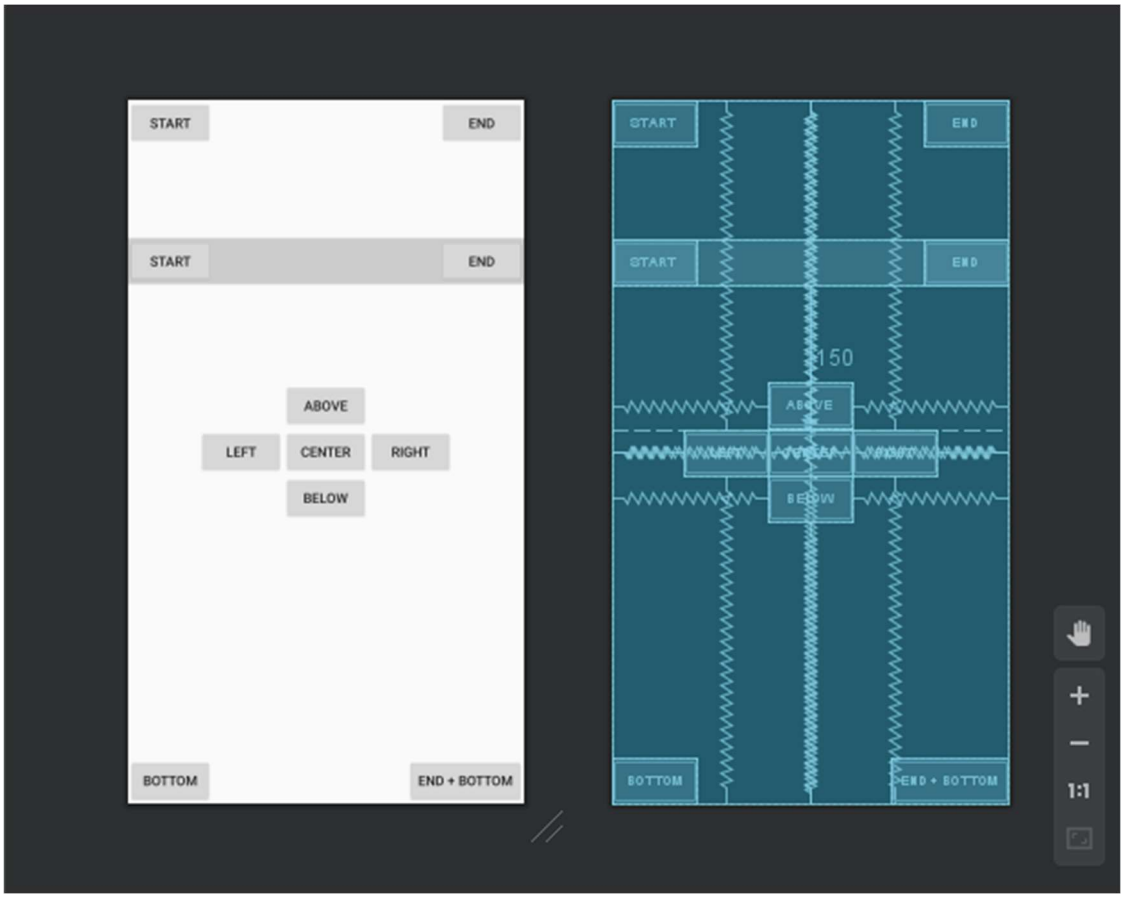
3.6. android:layout_below: il widget viene posizionato sotto al widget di cui verrà inserito l'id.

3.7. android:layout_toLeftOf: il widget viene posizionato a sinistra del widget di cui verrà inserito l'id.

3.8. android:layout_toRightOf: il widget viene posizionato a destra del widget di cui verrà inserito l'id.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="START"
        android:layout_alignParentStart="true"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="BOTTOM"
        android:layout_alignParentBottom="true"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="END"
        android:layout_alignParentEnd="true"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="END + BOTTOM"
        android:layout_alignParentEnd="true"
        android:layout_alignParentBottom="true"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="CENTER"
        android:id="@+id/center"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@id/center"
        android:layout_centerInParent="true"
        android:text="ABOVE" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:layout_toLeftOf="@id/center"
        android:text="LEFT" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:layout_toRightOf="@id/center"
        android:text="RIGHT" />
```

```
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/center"
        android:layout_centerInParent="true"
        android:text="BELOW" />

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ccc"
        android:layout_above="@id/center"
        android:layout_marginBottom="150dp">

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="START"
            android:layout_alignParentStart="true"/>

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="END"
            android:layout_alignParentEnd="true"/>

    </RelativeLayout>

</RelativeLayout>
```



4. Il file Manifest.xml.

Il file Manifest è un file XML che contiene informazioni dettagliate riguardo l'applicazione, più precisamente definisce:

- il nome del package;
- il codice della versione;
- il nome della versione;
- la versione del SDK minima per l'utilizzo dell'applicazione (minSDKVersion);
- l'immagine che rappresenta l'applicazione sul dispositivo (icon);
- il nome dell'applicazione (label);
- il nome delle activity presenti nell'applicazione.

Un nodo molto importante del file manifest.xml è il nodo <intent-filter> figlio del nodo <activity>. Questo nodo ha il preciso compito d'informare Android che la nostra Activity è in **grado di "svegliarsi"** in seguito al verificarsi di particolari messaggi/azioni trasmessi dall'interno del dispositivo o da altre applicazioni.
Il messaggio/azione a cui deve rispondere **OBBLIGATORIAMENTE** ogni nostra applicazione è proprio il lancio e l'avvio di questa:

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Per ogni nodo <intent-filter> che viene creato, è possibile specificare **3 tipologie di attributi**: ACTION, CATEGORY, DATA:

- <action>: qui si inserisce il **nome dell'azione** che verrà "ascoltata" dalla nostra Activity e che la nostra Activity sarà in grado di gestire;
- <category> (*opzionale): è un'ulteriore informazione che identifica il **tipo di azione**.
- <data> (*opzionale): fornisce informazioni sulla **tipologia di dati** eventuali che l'activity è in grado di **gestire** (testo, immagine, etc…) nel formato mime (es. text/plain).

5. Sistema multilingua di Android.

Stringe, array, e tanti altri tipi di dati che utilizziamo all'interno dell'applicazione possono essere dichiarate all'interno di una risorsa XML nella cartella res/values/.

In questo modo possiamo "mappare" qualsiasi stringa che compare nell'applicazione all'interno delle risorse res/values. La cartella values identifica la lingua di default dell'applicazione. Quando un'applicazione viene avviata, la piattaforma Android individua la lingua con cui è impostato il dispositivo ed in base ad essa carica la cartella values-xx dove xx è il suffisso della lingua. Se ad esempio in un'applicazione abbiamo definito una serie di stringhe all'intero della risorsa res/values/string.xml e vogliamo avere il supporto per la lingua italiana, inglese e francese, basta tradurre le stringhe di questo file e inserirle rispettivamente nelle sottocartelle:

- res/values/strings.xml: default (ad esempio inglese),
- res/values-**it**/strings.xml: lingua italiana,
- res/values-**fr**/strings.xml: lingua francese,
- res/values-**xx**/strings.xml: la lingua corrispondente al suffisso xx.

Se non viene individuata la sottocartella values-xx relativa alla lingua del dispositivo, allora viene caricata la cartella di default **values.**

6. Dimensioni in Android.

Android permette al programmatore di specificare le dimensioni usando varie unità di misura:

- dp (density-independent pixels)
- sp (scale-independent pixels)
- pt, points (1/72 di pollice)
- px, pixel reali
- mm, millimetri
- in, pollici (inches)

Vengono utilizzate diverse unità di misura per adattare i widget a diversi dispositivi. Dp, sp, pt sono screen density ovvero dipendono da quanti pixel ci sono nel display (dpi), in particolare tra dp e sp l'unica differenza è che gli sp scalano in base alla preferenza dell'utente sulla grandezza del font.

Px, mm, in rappresentano la grandezza reale del display.

L'unita dpi dipende dal rapporto tra risoluzione e dimensione dello schermo. Si tratta quindi del numero di pixel presenti in un pollice.

Ad esempio, pensiamo a due display di eguale grandezza, il primo con risoluzione 320*480 pixel e il secondo 480*800 pixel; il primo avrà un numero di DPI inferiori al secondo.

I dp nascono per semplificare la vita agli sviluppatori, in quanto permette di mantenere le proporzioni delle interfacce e non solo, ad esempio:

prendiamo un'icona di grandezza 50 pixel, e proviamo a inserirla prima su un display con 160 DPI e in seguito su un display di 240 DPI. Nel primo caso l'icona sarà più grande mentre nel secondo più piccola. Questo avviene perché lo schermo con maggiore densità avrà pixel più ravvicinati. Per ovviare a questo problema ed ottenere delle interfacce che siano uguali su tutti i device, si utilizzano i DP, come dice il nome sono "indipendenti". Questo significa che: 1 DP è uguale ad 1 pixel su uno schermo di 160 DPI. Per tutte le altre densità i pixel verranno scalati seguendo questa formula: 1 pixel = dp * (dpi / 160);

In questa maniera sarà possibile non riscrivere applicazioni per tutti i

• Adapter fornisce gli elementi da visualizzare in base allo scorrimento effettuato dall'utente

tipi di device esistenti.
Questo però non vale per le immagini, le quali non scalano automaticamente.

7. ListView e Adapter.

L'adapter è un componente collegato ad una struttura dati di oggetti Java (array, Collections, etc…) e che incapsula il meccanismo di trasformazioni di questi oggetti in altrettante View da mostrare su layout.
Una ListView non è nient'altro che una lista di view. Le view che compongono questa lista possono essere semplici view come una lista di TextView, oppure una lista di view più complesse.
Vediamo il codice per una ListView semplice:
Per prima cosa, dobbiamo specificare un file XML che descrive il layout di ogni singolo elemento appartenente alla ListView. In questo caso la singola riga sarà composta da una semplice TextView:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/textViewList"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="" android:padding="10dp"
        android:textSize="50dp"/>

</LinearLayout>
```

• Un listView è un Widget specifico per le liste che divide l'area disponibile in varie posizioni.
Il numero dipende dall'area disponibile e dalla grandezza di ogni elemento
• Gli elementi vengono memorizzati in un array
– solitamente sono di più rispetto alle posizioni disponibili nel widget
– Si può "scorrere" la lista

Successivamente dobbiamo inserire il widget ListView all'interno del layout che la ospiterà:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:layout_centerHorizontal="true"
        android:background="#CCDDEE"
        android:layout_marginTop="100dp"
        android:layout_width="300dp"
        android:layout_height="500dp">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical" >

            <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:textAppearance="?android:attr/textAppearanceLarge"
                android:gravity="center"
                android:textSize="50dp"
                android:background="#FFDDEE"
                android:text="ListView"/>

            <ListView
                android:id="@+id/mylistview"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" >
            </ListView>

        </LinearLayout>
    </FrameLayout>
</RelativeLayout>
```

Ora non ci resta che creare un oggetto di tipo ArrayAdapter, che ci servirà per adattare ogni singolo oggetto in una riga da aggiungerà alla ListView:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);


    String [] array = {"Pasquale","Maria","Michele","Antonella", "Vincenzo",
            "Teresa", "Roberto", "Rossella", "Antonio", "Luca", "Liliana", "Stefania",
            "Francesca", "Andrea", "Marco", "Elisa", "Anna", "Lorenzo"};

    listView = findViewById(R.id.mylistview);

    ArrayAdapter<String> arrayAdapter =
            new ArrayAdapter<String>(this, R.layout.list_element, R.id.textViewList,
array);

    listView.setAdapter(arrayAdapter);

    listView.setOnItemClickListener(new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long
id) {
            String  str  = listView.getItemAtPosition(position).toString();
            Toast.makeText(getApplicationContext(),
                    "Click su posizione n." + position + ": " + str, Toast.LENGTH_LONG)
                    .show();
        }
    });
}
```

Come si vede dal codice, i passi da seguire sono:
- creiamo un'istanza dell'oggetto ArrayAdapter passando al costruttore il layout che descrive com'è fatta ogni singola riga della ListView, e l'id della View che verrà adattata con l'oggetto da visualizzare.
- settiamo l'adapter della ListView con il metodo setAdapter passando come argomento il nostro arrayAdapter.
- infine, se necessario, possiamo settare un listener sul click di ogni riga della ListView.

È possibile creare ListView customizzate per poter rappresentare oggetti più complicati. Ad esempio: vogliamo visualizzare per ogni riga della ListView, un nome, un numero e un'immagine.
Come prima cosa, definiamo il layout di ogni singola riga del file XML:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >

    <ImageButton
        android:id="@+id/elem_lista_foto"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:onClick="onPictureClick"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

    <Button
        android:id="@+id/elem_lista_nome"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="left|center_vertical"
        android:onClick="onNameClick"
        style="?android:attr/borderlessButtonStyle"
        android:textSize="22dp"/>

    <Button
        android:id="@+id/elem_lista_telefono"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="onTelClick"
        android:gravity="left"
        style="?android:attr/borderlessButtonStyle"
        android:textSize="12dp"/>

    </LinearLayout>
</LinearLayout>
```

Come possiamo notare, ogni attributo possiede il tag onClick, in questo modo andremo a gestire una ListView dove ogni componente di una singola riga chiama un metodo diverso.

Ora non ci resta che definire il layout dell'activity che ospiterà la nostra ListView cutomizzata.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="match_parent"
 android:layout_height="match_parent">

 <FrameLayout
   android:layout_centerHorizontal="true"
   android:background="#CCDDEE"
   android:layout_marginTop="100dp"
   android:layout_width="500dp"
   android:layout_height="700dp">

   <LinearLayout
     android:layout_width="match_parent"
    android:layout_height="wrap_content"
     android:orientation="vertical" >

     <TextView
       android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:textAppearance="?android:attr/textAppearanceLarge"
      android:gravity="center"
      android:background="#FFDDEE"
      android:text="Custom ListView MultiClick"/>

     <ListView
         android:id="@+id/mylistview"
         android:layout_width="match_parent"
         android:layout_height="wrap_content" >
     </ListView>

   </LinearLayout>
 </FrameLayout>
</RelativeLayout>
```

Ora passiamo al codice sorgente. Andiamo a definire la class CustomAdapter che estenderà la classe ArrayAdatper <Contatto>

```
public class CustomAdapter extends ArrayAdapter<Contatto> {
    private LayoutInflater inflater;

    public CustomAdapter(Context context, int resourceId, List<Contatto> objects) {
            super(context, resourceId, objects);
            inflater = LayoutInflater.from(context);
    }

    @Override
    public View getView(int position, View v, ViewGroup parent) {
          if (v == null) {
             Log.d("DEBUG","Inflating view");
             v = inflater.inflate(R.layout.list_element, null);
          }

            Contatto c = getItem(position);

         Log.d("DEBUG","contact c="+c);

            Button nameButton;
            Button telButton;
            ImageButton fotoButton;

            nameButton = (Button) v.findViewById(R.id.elem_lista_nome);
            telButton = (Button) v.findViewById(R.id.elem_lista_telefono);
            fotoButton = (ImageButton) v.findViewById(R.id.elem_lista_foto);

            fotoButton.setImageDrawable(c.getPicture());
            nameButton.setText(c.getName());
            telButton.setText(c.getTel());

            fotoButton.setTag(position);
            nameButton.setTag(position);
            telButton.setTag(position);

            return v;
    }
}
```

Come possiamo notare, dobbiamo fare l'override del metodo getView della classe ArrayAdapter. Questo metodo serve per trasformare l'oggetto Contatto in una riga della ListView.
Il metodo getView prende input diversi argomenti: position (che indica la posizione dell'elemento all'interno della ListView), v e parent.
Se la view è null, allora abbiamo bisogno di inizializzarla con il riferimento al file XML che descrive il layout di ogni singola riga del layout.

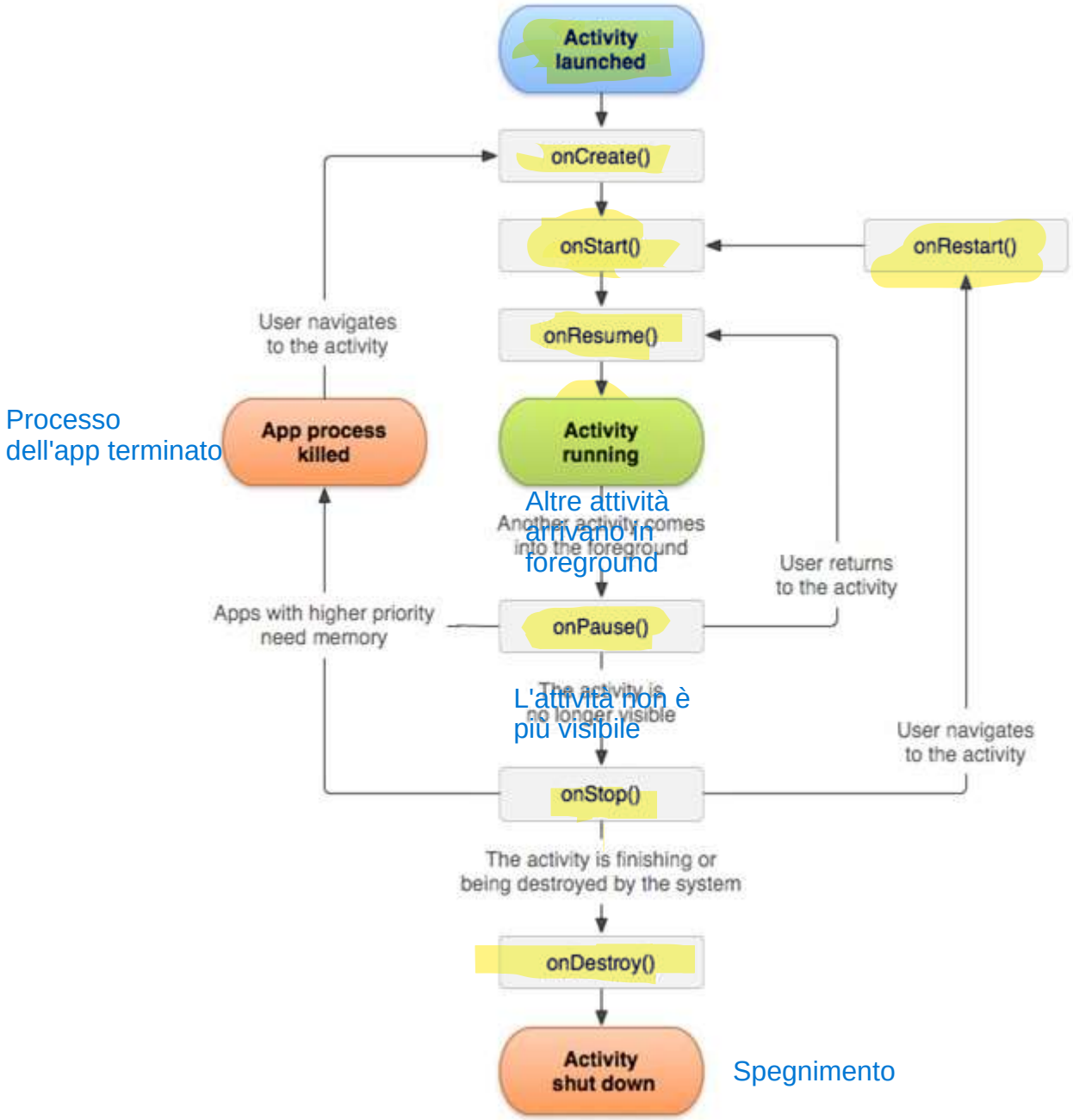
Ora che la view è stata inizializzata, possiamo usare il parametro position, per ottenere l'oggetto Contatto che ci interessa dalla collezione di oggetti che ci è stata assegnata dal metodo costruttore.
Ora non ci resta che fare delle operazioni molto semplici: otteniamo tutti i riferimenti ai widget che ci interessano utilizzando il metodo findViewById, e settiamoli in modo tale che rappresentino tutte le informazioni che a noi interessano.
Una volta settato il nome, numero e l'immagine del contatto, possiamo settare come tag di ogni widget la position dell'oggetto nella ListView. Questo può essere molto utile perché potrebbe servirci conoscere la posizione dell'oggetto nella ListView. In precedenza, questa operazione non veniva effettuata perché il click su ogni elemento veniva gestito con un Listener di tipo OnItemClickListener che implementa il metodo onClick che, tra i vari argomenti che prendi in input, c'era proprio l'argomento position. Poiché per come è stato progettato il layout non viene più gestita una singola operazione per ogni riga, bensì 3, i metodi che andiamo a implementare non avranno come argomento in input "position", e quindi potremmo aver bisogno di ricavarla in qualche modo; così facendo ci basterà fare View.getTag();

Andiamo quindi ora ad implementare la classe Java dell'activity che ospiterà la ListView.

```
public class MainActivity extends Activity {
   public ListView listView;
   CustomAdapter customAdapter;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      String[] nomi = {"Pasquale","Maria","Michele","Antonella", "Vincenzo",
            "Teresa", "Roberto", "Rossella", "Antonio", "Luca"};

        listView = (ListView)findViewById(R.id.mylistview);

      customAdapter = new CustomAdapter(this, R.layout.list_element, new
ArrayList<Contatto>());

        listView.setAdapter(customAdapter);

        for (int i=0; i<nomi.length; i++) {
           Contatto c = new Contatto(
                 nomi[i],
                 "111-2222-333",
                 getResources().getDrawable(R.drawable.faceplaceholder));
           customAdapter.add(c);
        }

   }

   public void onPictureClick(View v) {
      int position = Integer.parseInt(v.getTag().toString());
      Contatto c = customAdapter.getItem(position);
       Toast.makeText(getApplicationContext(),
                  "Click su Foto - posizione n."+position+": " +c.getName(),
Toast.LENGTH_LONG)
                  .show();
   }

   public void onNameClick(View v) {
      int position = Integer.parseInt(v.getTag().toString());
      Contatto c = customAdapter.getItem(position);
       Toast.makeText(getApplicationContext(),
                  "Click su Nome - posizione n."+position+": " +c.getName(),
Toast.LENGTH_LONG)
                  .show();
   }

   public void onTelClick(View v) {
      int position = Integer.parseInt(v.getTag().toString());
      Contatto c = customAdapter.getItem(position);
       Toast.makeText(getApplicationContext(),
                  "Click su Tel - posizione n."+position+": " +c.getName(),
Toast.LENGTH_LONG)
                  .show();   }

}
```

Le operazioni che vengono effettuate sono molto semplice:
- viene prelevato il riferimento alla ListView;
- viene istanziato il customer adapter;
- si usa il metodo setAdapter per collegare il customer adapter alla ListView;
- si implementano i 3 metodi onPictureClick, onNameClick, onTelClick.

8. Ciclo di vita di un'activity.

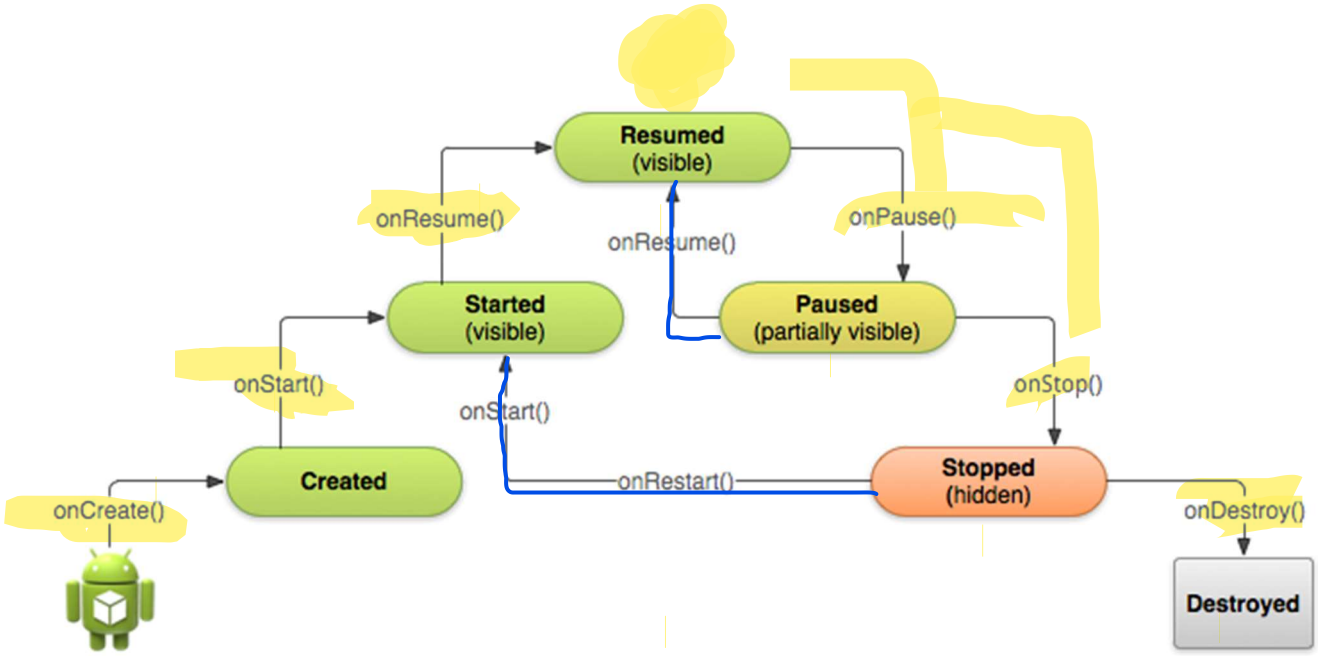


Processo dell'app terminato

Altre attività arrivano in foreground

L'attività non è più visibile

Spegnimento



Quando viene avviata un'activity vengono lanciati i metodi: onCreate() -> onStart() -> onResume().
A questo punto un'activity si trova nello stato di Resumed.
Quando l'applicazione viene chiusa vengono lanciati i metodi: onPause() -> onStop() -> onDestroy().
Se l'utente sta utilizzando l'activity (stato Resumed) e decide di premere il tasto Home, l'activity passa dallo stato di Resumed allo stato di Stopped. Vengono chiamati i metodi: onPause() -> onStop().
Quando l'utente decide di ritornare all'activity, essa passa dallo stato di Stopped allo stato di Resumed. Vengono chiamati i metodi: onRestart() -> onStart() -> onResume(). Da notare che il metodo onRestart chiama sempre il metodo onStart, quindi molto spesso le operazioni di onRestart vengono effettuate direttamente nel metodo onStart.
Quando l'utente ruota il dispositivo, l'activity viene prima distrutta e poi ricreata. Vengono quindi lanciati i metodi: onPause() -> onStop() -> onDestroy() -> onCreate() -> onStart() -> onResume(). Questo perché così facendo l'activity può adattarsi al meglio alla nuova configurazione.
Questo però causa un grande problema: il metodo onDestroy distrugge l'activity provando quindi anche la perdita del suo stato. È quindi necessario sovrascrivere il metodo

onSaveInstanceState(Bundle savedInstanceState) per poter salvare lo stato della nostra activity prima che essa venga distrutta.
Questo metodo viene lanciato automaticamente prima del metodo onDestroy. Esso prende in input un oggetto di tipo Bundle: questo oggetto ci da la possibilità di poter salvare delle coppie chiave/valori che poi verranno riprese nel metodo onCreate.
Un esempio di codice:

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
      savedInstanceState.putStringArrayList("LISTA_CHIAMATE", array);
      savedInstanceState.putInt("COUNTER", counter);
      savedInstanceState.putString("EDIT_TEXT", str);
      super.onSaveInstanceState(savedInstanceState);
}
```

NB: Dopo aver inserito i vari oggetti all'interno dell'oggetto Bundle, dobbiamo chiamare il metodo onSaveInstanceState del superclasse passandogli l'oggetto Bundle come argomento.

A questo punto, ci basterà gestire l'oggetto Bundle nel metodo onCreate per ottenere il vecchio stato e poterlo ripristinare.
La prima cosa da fare è controllare se l'oggetto Bundle è diverso da null. In questo caso vuol dire che esiste uno stato da ripristinare, mentre, nel caso in cui l'oggetto Bundle è uguale a null, vuol dire che non vi è nessuno stato da ripristinare.
A questo punto non ci basterà che utilizzare i vari metodi get per poter ottenere i valori che interessano dall'oggetto Bundle.

```
if (savedInstanceState != null) {
      array = savedInstanceState.getStringArrayList("LISTA_CHIAMATE");
      counter = savedInstanceState.getInt("COUNTER");
      str = savedInstanceState.getString("EDIT_TEXT");
}
```

Android ci permette di gestire in proprio il cambiamento. Così facendo l'activity NON viene più distrutta ma viene eseguito il metodo onConfigurationChanged().
Per poter gestire in proprio il cambiamento ci basterà scrivere nel file manifest.xml:

```
<activity android:name=".MainActivity"
      android:configChanges="orientation">
```

Ora non ci resta che sovrascrivere il metodo onConfigurationChanged() per poter gestire in proprio i vari cambi di orientamento dello schermo.

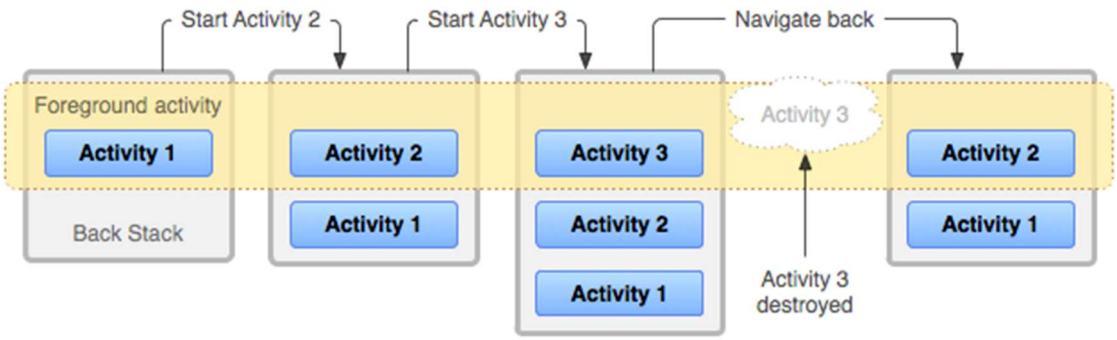
```
@Override
public void onConfigurationChanged(Configuration newConfig) {
      super.onConfigurationChanged(newConfig);

      // Checks the orientation of the screen
      if (newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {
            Toast.makeText(this, "landscape", Toast.LENGTH_SHORT).show();
      } else if (newConfig.orientation == Configuration.ORIENTATION_PORTRAIT){
            Toast.makeText(this, "portrait", Toast.LENGTH_SHORT).show();
      }
}
```

Anche se Android ci permette di gestire in proprio il cambiamento, questa tecnica è altamente sconsigliata.

9. Backstack

Un'applicazione è fatta di più activity

Più activity possono coesistere tra loro e sono organizzate in un backstack. Esempio: solitamente un task (insieme di Activity con le quali l'utente interagisce) parte dalla Home. Quando l'utente clicca su un'icona e lancia una nuova activity, l'app è mostrata a schermo (foreground). Nel momento in cui vengono lanciate nuove activity, la corrente è messa nel backstack e ci può tornare col pulsante back. Un task con le sue activity può essere spostato in background (quando l'utente inizia un task o clicca sull'home per esempio) e quindi le sue activity verranno messe in stato Stopped, ma il loro backstack rimane intatto.
Il backstack considera solo le activity, per i fragments dobbiamo gestirli manualmente. Per inserire i cambiamenti nel backstack bisogna creare un metodo addToBackStack(). Nel caso in cui non chiameremo il metodo addToBackStack() i cambiamenti andranno persi.



Supponiamo che sia in esecuzione l'activity A. Essa lancia una nuova activity B che a sua volta lancia l'activity C che a sua volta lancia l'activity D. Cosa si deve fare se si vuole fare in modo che dall'activity D si torni direttamente ad A quando si preme il pulsante di back?

Ci sono due metodi:

- Possiamo sovrascrivere il metodo onBackPressed, in cui dichiariamo un intent che lancia l'activity A.

```
@Override
public void onBackPressed() {
    Intent i = new Intent();
    i.setClass(getApplicationContext(), ActivityA.class);
    startActivity(i);
}
```

- Possiamo settare un flag agli intent che lanciano l'activity B e C di tipo Intent.FLAG_ACTIVITY_NO_HISTORY. Così facendo, quando verrà chiamata l'activity B e l'activity C, queste non verranno inserite nel backstack. Così facendo, quando nell'activity D verrà premuto il tasto back, torneremo all'activity A poiché sarà l'unica activity presente all'interno del backstack.

  Vediamo il codice delle quattro activity:

```
public class ActivityA extends AppCompatActivity {

    ...

    public void next(View v) {
        Intent i = new Intent();
        i.setClass(getApplicationContext(), ActivityB.class);
        i.setFlags(Intent.FLAG_ACTIVITY_NO_HISTORY);
        startActivity(i);
    }

}
```

```
public class ActivityB extends AppCompatActivity {

    ...

    public void next(View v) {
        Intent i = new Intent();
        i.setClass(getApplicationContext(), ActivityC.class);
        i.setFlags(Intent.FLAG_ACTIVITY_NO_HISTORY);
        startActivity(i);
    }

}
```

```
public class ActivityC extends AppCompatActivity {

    ...

    public void next(View v) {
        Intent i = new Intent();
        i.setClass(getApplicationContext(), ActivityD.class);
        startActivity(i);
    }

}
```

10. Intent Vedi slide

Gli intent possono essere:

- espliciti: viene dichiarato quale componente dovrà essere attivata. Particolarmente utili nell'apertura di una nuova Activity.
- impliciti: non specificano una componente da attivare ma quale azione deve essere svolta. La loro invocazione si estrinseca spesso nell'apertura di una finestra di dialogo che chiede all'utente quale app vuole si apra per completare l'azione. Android scegli un'attività appropriata, in base all'action, type, uri e category. Le attività dichiarano le action che possono soddisfare nel manifesto.

Negli Intent è possibile aggiungere degli "extra", ovvero delle coppie chiave-valore che possono essere recuperate dalla componente che viene attivata. Facciamo degli esempi.

Ipotizziamo che un activity A debba lanciare un activity B passandogli una stringa ed un intero e che attenda un risultato dall'activity B di tipo booleano. Nell'activity A dobbiamo quindi creare un intent ed inserire come extra la stringa e l'intero che vogliamo inviare all'activity B. Infine dobbiamo usare il metodo startActivityForResult. In questo modo, non appena l'activity B

tornerà all'activity A, verrà invocato il metodo onActivityResult. In questo metodo non dobbiamo fare nient'altro che ottenere il nostro risultato, ovvero prelevare l'oggetto boolean contenuto nell'intent data (argomento passato in un input).

```
public void startActivityB(View v) {
       Intent i = new Intent();
       i.setClass(getApplicationContext(), ActivityB.class);
       i.putExtra("stringa", "stringa");
       i.putExtra("intero", 10);
       startActivityForResult(i, 0);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode != 0) return;
    if (resultCode != Activity.RESULT_OK) return;
    if (data == null) return;
    result = data.getBooleanExtra("risultato", false);

}
```

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

    ...

    Intent i = getIntent();
    stringa = i.getStringExtra("stringa");
    intero = i.getBooleanExtra("intero", 0);
}

private void setReturnIntent() {
    Intent data = new Intent();
    data.putExtra("risultato", true);
    setResult(RESULT_OK,data);
}
```

Questo invece è un esempio di codice dove l'activity lancia un intent di tipo implicito.

```
public void visualizzaMappaClicked(View v) {
    String address = indirizzo.getText().toString();
    if (null != address) {
        address = address.replace(' ', '+');
        Intent geoIntent = new Intent(android.content.Intent.ACTION_VIEW,
                Uri.parse("geo:0,0?q=" + address));
        startActivity(geoIntent);
    }
}
```

11. Permessi

Android protegge risorse e dati con un meccanismo di permessi di accesso. Questo serve per limitare l'accesso a informazioni dell'utente, servizi con costi, risorse di sistema etc…
I permessi sono divisi in due classi: normali e pericolosi.
- I permessi normali sono quelli che non vanno ad interferire con la privacy dell'utente, per questo motivo vengono concessi senza chiedere nulla all'utente. Un esempio di questo è l'accesso alla rete (android.permission.INTERNET).
- I permessi pericolosi, invece, sono potenzialmente più lesive della riservatezza degli utenti. Quest'ultimi devono essere approvati dall'utente quando si installa l'app (API < 23) o a runtime (API >= 23). Quando l'app richiede un permesso pericoloso, se ha già un permesso per lo stesso gruppo viene concesso automaticamente, altrimenti viene richiesto all'utente, tramite un dialog box, il permesso per il GRUPPO.
I permessi vengono rappresentati da stringhe. Ogni app deve dichiarare nel manifesto i "permessi" che intende utilizzare.

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Quando si installa un app bisogna accettare i permessi, inoltre, quando un'app viene lanciata avremo la possibilità di rendere definitiva la risposta. All'interno delle impostazioni nella sezione app installate->autorizzazioni è possibile cambiare la decisione e non accettare più quel determinato permesso pericoloso.

I thread permettono la computazione parallela all'interno di un processo
- ogni thread ha il proprio program counter ed il proprio stack
- condivide con gli altri thread del processo l'heap e la memoria statica

Implementano l'interfaccia Runnable e devono avere il metodo void run()

12. Thread

Quando si utilizza un'activity, il thread principale dell'applicazione si occupa prevalentemente di gestire i messaggi relativi al funzionamento dell'interfacci utente. Svolgere in questo stesso thread operazioni presumibilmente "lente" come, ad esempio, la lettura e scrittura da file, il prelevamento di dati da un database, il caricamento di immagini, rischierebbe di rendere poco reattiva la UI. Conseguenza di ciò sarebbe una user experience non troppo gradevole. Quindi, operazioni "lente" dovrebbero essere preferibilmente svolte su thread secondari. Inoltre, l'accesso in rete deve obbligatoriamente essere eseguito su un thread secondario.
C'è solo un problema: in Android non è possibile modificare l'interfaccia utente da un thread secondario. Per questo motivo c'è bisogno di un sistema di comunicazione tra il thread secondario e il thread principale. Per questo motivo esiste la classe AsyncTask.
I metodi contenuti in questa classe sono di due tipi, metodi che vengono eseguiti dal thread principale e un solo metodo che viene eseguito dal thread secondario. Tra i vari metodi vi sono:

- onPreExecute: che serve per inizializzare le operazioni prima che avvenga l'esecuzione del thread secondario. Questo metodo viene eseguito dal thread main.
- doInBackground: questo metodo viene invocato dal thread secondario. All'interno di questo metodo andremo a collocare tutte le operazioni "lente".
- onProgressUpdate: viene invocato ogni vota che dall'interno di doInBackground viene chiamato il metodo publishProgress. Serve a fornire aggiornamenti periodici all'interfaccia utente.
- onPostExecute: viene eseguito alla fine di doInBackground ed anch'esso svolge operazioni collegate all'interfaccia utente.

Un aspetto importante della classe AsyncTask cui si deve presentare attenzione sono i parametri della classe: AsyncTask <Type1, Type2, Type3>. Questi tre tipi di dato saranno, rispettivamente, il tipo di dato accettato in input dai metodi doInBackground,

• Classe Java generica class AsyncTask<Params, Progress, Result> {}
• Parametri
– Params: tipo di dati per il lavoro che deve svolgere il background thread
– Progress: tipo di dati usato per lo stato di avanzamento
– Result: tipo di dati per il risultato del task

onProgressUpdate e onPostExecute.

Vediamo un esempio di codice:

```
class LoadIconTask extends AsyncTask <Integer, Integer, Bitmap> {

   @Override
   protected void onPreExecute() {
      progressBar.setVisibility(ProgressBar.VISIBLE);
   }

   @Override
   protected Bitmap doInBackground(Integer... img_ids) {
      Bitmap tmp = BitmapFactory.decodeResource(getResources(), img_ids[0]);

      for (int i = 1; i < 11; i++) {
         sleep();
         publishProgress(i * 10);
      }

      return tmp;
   }

   @Override
   protected void onProgressUpdate(Integer... values) {
      progressBar.setVisibility(ProgressBar.VISIBLE);
      progressBar.setProgress(values[0]);
      if (values[0]>75) {
         Toast.makeText(getApplicationContext(),"Stiamo per terminare",
                Toast.LENGTH_LONG).show();
      }
   }

   @Override
   protected void onPostExecute(Bitmap result) {
      progressBar.setVisibility(ProgressBar.INVISIBLE);
      progressBar.setProgress(0);
      imageView.setImageBitmap(result);
   }

   private void sleep() {
      try {
         Thread.sleep(1000);
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
   }
}
```

13. Fragments Vedi slide

Un fragment è una porzione di activity. Non si tratta solo di un gruppo di controlli o di una sezione del layout. Può essere definito come una specie di sub-activity con un suo ruolo funzionale molto importante ed un suo ciclo di vita.
Un'activity può ospitare vari frammenti che possono essere inseriti e rimossi durante l'esecuzione. Si possono creare UI con molti frammenti anche in funzione della grandezza dello schermo.
Un fragment ha il suo ciclo di vita fortemente collegato con quello dell'activity di appartenenza: se l'activity è in stato "paused" lo sono anche tutti suoi frammenti.
Per poter operare con i frammenti vi è la necessita di un manager che possiamo ottenere con il metodo getFragmentManager().
A questo punto possiamo effettuare delle operazioni di transazione sui fragment. Le operazioni più comuni sono: add, remove, replace, hide e show. Come avviene per le transazioni nei database, le operazioni iniziano con un beginTransaction e vengono definitivamente salvate con un commit. Per fare ciò abbiamo bisogno di un riferimento all'oggetto FragmentTransaction che possiamo ottenere con il metodo beginTransaction dell'oggetto FragmentManager.
Cosa molto importante, il backstack considera solo le activity, quindi per inserire i cambiamenti nel backstack anche per i frammenti bisogna usare il metodo addToBackStack(). Se non chiamiamo addToBackStack, quando premiamo back "salteremo" i cambiamenti fatti con i frammenti.
I frammenti possono comunicare con l'activity creando dei metodi di callback.

NETWORKING SALTATA

14. Data Storage

Vi sono tre metodi diversi per rendere i dati persistenti in Android:

- SharedPreferences: la classe SharedPreferences permette di salvare e recuperare dati usando coppie di chiave/valore. I dati vengono salvati in un file XML contenuto nello storage intero, precisamente nella cartella shared_prefs. E' possibile ottenere un riferimento alla classe SharedPreferences in due modi: con il metodo getDefaultSharedPreferences() quando basta un solo file, oppure con il metodo getSharedPreferences("filename"), quando si vogliono usare più file. **Attenzione**. Non usare il metodo getPreferences, perché così facendo si ottiene sempre un oggetto di tipo SharedPreferences, ma in questo caso i dati non verranno condivise con altre activity dell'app.

  Questo oggetto contiene i tipici metodi di lettura come contains(String key) oppure i classi getter come getInt, getString, getFloat etc… Per poter invece inserire delle preferenze invece, bisogna utilizzare un oggetto di tipo Editor. Per ottenere un riferimento a quest'ultimo ci basterà utilizzare il metodo edit() dell'oggetto SharedPreferences. Una volta ottenuto l'oggetto Editor, possiamo utilizzare i classici metodi di put per poter inserire i nostri dati all'interno del file XML (Editor.putBoolean, Editor.putInt, etc…). Una volta inseriti i dati, bisogna fare una commit per rendere i dati persistenti chiamando semplicemente il metodo commit dell'oggetto Editor.

- File: ogni app ha a disposizione uno spazio disco, detto Storage interno che risiede in una parte del filesystem e a cui solo l'applicazione dovrebbe accedere, inoltre, se l'app viene disinstallata, anche la directory viene cancellata. Per poter aprire un file in lettura si usa il metodo openFileInput(String filename) che restituisce un oggetto di tipo FileInputStream, mentre per aprire uno stream in scrittura ci basterà usare il metodo openFileOutput(String filename, int mode) che ci restituirà un oggetto di tipo FileOutputStream. Oltre allo storage interno, l'applicazione può accedere anche allo

storage esterno. L'accesso a quest'ultimo avviene mediante la classe Environment. La prima operazione da svolgere è controllare lo stato del supporto. Lo si fa con il metodo statico Environment.getExternalStorageState() che restituisce una stringa che verrà confrontata con le costanti della classe Environment. La più importante è Environment.MEDIA_MOUNTED che indica che il support è disponibile in lettura e scrittura.
- SQL: Android fornisce supporto per database SQL solo all'interno dell'app. Per usare un database bisogna creare una sottoclasse di SQLiteOpenHelper che servirà per gestire la nascita e l'aggiornamento del database su memoria fisica e recuperare un riferimento all'oggetto SQLiteDatabase, usato come accesso ai dati. Una classe molto importante è la classe Cursor. Quest'ultima rappresenta un puntatore ad un set di risultati della query. Un oggetto Cursor può essere spostato per puntare ad una riga differente del set di risultati.

15. Animazioni Slide

Android permette di definire delle animazioni da applicare alle immagini descrivendole tramite file XML o programmaticamente. Le animazioni disponibili sono rotation, translate, scale, alpha…
Per descrivere un'animazione con il file XML bisogna creare il file all'interno della directory res/anim. Il file XML ha come radice il tag <set> che racchiude tutte le operazioni che verranno effettuate per quell'animazione.

Uno snippet di codice:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">

    <translate
        android:startOffset="0"
        android:duration="1500"
        android:fromXDelta="0"
        android:fromYDelta="0"
        android:interpolator="@android:anim/linear_interpolator"
        android:toXDelta="150"
        android:toYDelta="150" />

    <scale
        android:startOffset="1500"
        android:duration="3000"
        android:fromXScale="1"
        android:fromYScale="1"
        android:toXScale="0.1"
        android:toYScale="0.1"
        android:pivotX="50%"
        android:pivotY="50%" />

    <rotate
        android:startOffset="1500"
        android:duration="3000"
        android:fromDegrees="0"
        android:toDegrees="720"
        android:interpolator="@android:anim/accelerate_interpolator"
        android:pivotX="50%"
        android:pivotY="50%" />

</set>
```

Nei primi 1,5 secondi l'immagine viene spostata di 150 pixel in basso e 150 pixel a destra contemporaneamente, in modo da formare uno spostamento obbliquo. L'animazione avviene in modo lineare, quindi sempre con la stessa velocità.
Nei successivi 3 secondi, l'immagine viene scalata, del 90% della sua dimensione mentre ruota su sé stessa per 3 volte. Durante questi 3 secondi l'immagine torna nella sua posizione originale. Le 3 rotazioni non avvengono con la stessa velocità, durante i 3 secondi in cui l'immagine ruota, si ha un effetto di accelerazione, quindi la prima rotazione sarà più lenta rispetto alle altre due.
Una volta descritta, per poterla utilizzare, basterà utilizzare il metodo setAnimation del riferimento che vogliamo animare:

```
b.setAnimation(AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.animazione));
```

In Java, invece, le animazioni vengono create con l'ausilio della classe ViewPropertyAnimator. Questa classe offre le animazioni più comuni, che possono essere invocate mediante appositi metodi come rotation, scale, alpha etc…
Per ottenere un riferimento ad un oggetto ViewPropertyAnimator è necessario invocare il metodo animate() sulla view. Uno snippet di codice può essere:

```
b.animate().rotation(200);
```

Per far sì che la view rimanga nella posizione finale dell'animazione, basta usare il metodo setFillAfter(true);

```
Animation animation = AnimationUtils.loadAnimation(getApplicationContext(),
R.anim.animazione);
animation.setFillAfter(true);
b.setAnimation(animation);
```

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?

```
Animation animation = AnimationUtils.loadAnimation(getApplicationContext(),
R.anim.animazione);
animation.setAnimationListener(new Animation.AnimationListener() {

    @Override
    public void onAnimationStart(Animation animation) {

    }

    @Override
    public void onAnimationEnd(Animation animation) {
        parentView.removeViewAt(0);
    }

    @Override
    public void onAnimationRepeat(Animation animation) {

    }
});
b.setAnimation(animation);
```

16. Meccanismo di layout

L'albero delle view rappresenta la gerarchia delle view. Le fasi necessarie per la visualizzazione del layout sono tre:
- Fase di misurazione: nella fase di misurazione c'è un processo di negoziazione tra parent e figli I figli definiscono una measuredSize ovvero quanto grande la view dovrebbe essere, che poi si trasformerà nella size reale, ovvero quanto grande la view sarà in realtà. Ogni view può esprimere la propria preferenza usando la classe ViewGroup.LayoutParams. La misurazione avviene nel metodo onMeasure();

- Fase di layout: la View Parent decide la grandezza e la posizione dei figli in accordo alle misure fatte nella fase precedente. Questo avviene nel metodo onLayout();
- Fase di disegno: la view viene disegnata col metodo onDraw(). Quando c'è un cambiamento si utilizza invalidate() che chiama onDraw() su tutta la view e requestLayout() che ripete l'intero processo su tutto l'albero.

17. MotionEvent

Il multitouch permette il rilevamento di uno o più tocchi sullo schermo. Il tocco è rappresentato dal pointer (singolo evento) e il movimento dal MotionEvent.
Il MotionEvent rappresenta un movimento registrato da una periferica. Il movimento è rappresentato da un ACTION_CODE (cambiamento avvenuto) e ACTION_VALUES (posizione e proprietà del movimento).
Gli ACTION_CODE possono essere:
- ACTION_DOWN: un dito tocca lo schermo ed è il primo;
- ACTION_POINTER_DOWN: un dito tocca lo schermo ma non è il primo;
- ACTION_MOVE: un dito che è sullo schermo si muove.
- ACTION_POINTER_UP: un dito che è sullo schermo non lo tocca più.
- ACTION_UP: l'ultimo dito sullo schermo viene alzato.

18. Sensori

MEDIA PLAYER SALTATO

Molti smartphone, tablet hanno sensori di movimento, di ambiente, di posizione, etc… I sensori forniscono dati "grezzi" e l'accuratezza dei dati dipende dalla qualità.
Grazie al SensorManager, possiamo conoscere quali sensori sono disponibili e conoscere anche le caratteristiche per ognuno di essi.
Per ottenere un riferimento al SensorManager bisogna usare il metodo getSystemService(SENSOR_SERVICE); Questo ci permette di poter leggere i dati grezzi del sensore e usare listeners sui

cambiamenti di dati. Per ottenere un riferimento all'accelerometro, ad esempio, ci basterà fare:
sesnsorManager.getDefaultSensor(SENSOR.TYPE_ACCELEROMETER);
I listener devono essere registrati nel metodo onResume e rilasciati nel metodo onPause per evitare un consumo di batteria inutile.

```
@Override
protected void onResume() {
    super.onResume();
    sensorManager.registerListener(this, accelerometer,SensorManager.SENSOR_DELAY_UI);
}

@Override
protected void onPause() {
    sensorManager.unregisterListener(this);
    super.onPause();
}
```

Il metodo registerListener prende in input un intero samplingPeriodUs, ovvero una specifica sulla velocità di campionamento. Android fornisce dei valori predeterminati da usare:
- SENSOR_DELAY_NORMAL (0.2sec), consigliabile per il cambio di orientamento dello schermo;
- SENSOR_DELAY_GAME (0.02sec), consigliabile per i giochi;
- SENSOR_DELAY_UI (0.06sec) è consigliabile per l'interfaccia utente;
- SENSOR_FASTEST (0sec) permette di ottenere i dati dal sensore il più velocemente possibile.
Il vantaggio di avere una frequenza di campionamento maggiore è quella di avere dei dati più precisi, a discapito però del consumo della batteria.
Per gestire gli eventi, la classe deve implementare l'interfaccia SensorEventListener e quindi anche i suoi metodi.
Nel metodo onSensorChanged(SensorEvent event), possiamo gestire il cambiamento di stato del sensore tramite l'oggetto SensorEvent.

19. Toast customizzato

Un toast è la forma di notifica più immediata che esiste, spunta dal basso per breve tempo per inviare un feedback all'utente.

```java
public void showCustomToast(View v) {
    Toast toast = new Toast(getApplicationContext());
    toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
    toast.setDuration(Toast.LENGTH_LONG);
    toast.setView(getLayoutInflater().inflate(R.layout.custom_toast, null));
    toast.show();
}
```

20. Content Providers, Broadcast, Services

- Content Provider: nasce con lo scopo della condivisione di dati tra applicazioni. Questi componenti permettono di condividere, nell'ambito del sistema, contenuti custoditi in un database, su file o reperibili mediante accessi in rete.
- Broadcast Receiver: è un componente che reagisce ad un invio di messaggi a livello di sistema con cui Android notifica l'avvenimento di un determinato evento, ad esempio l'arrivo di un SMS. Questi componenti sono particolarmente utili per la gestione istantanea di determinate circostanze speciali.
- Service: svolge un ruolo opposto all'activity. Rappresenta un lavoro che viene svolto interamente in background senza bisogno di iterazione diretta con l'utente. I servici sono di due tipologie: started e bounded: un service è started quando un'app ha bisogno di svolgere attività in background, mirate ad uno scopo specifico, fino al loro completamento; i service bounded vengono attivati solo nel caso in cui un'altra app abbia bisogno di connettersi a loro.

Esempi:
Broadcast: invio messaggio
Service: posizione in tempo reale ogni 2 sec
Content provider: invio di informazioni come l'invio di arametri ad un'altra activity