Activity Life cycle

Programmazione Mobile

A.A. 2021/22

M.O. Spata

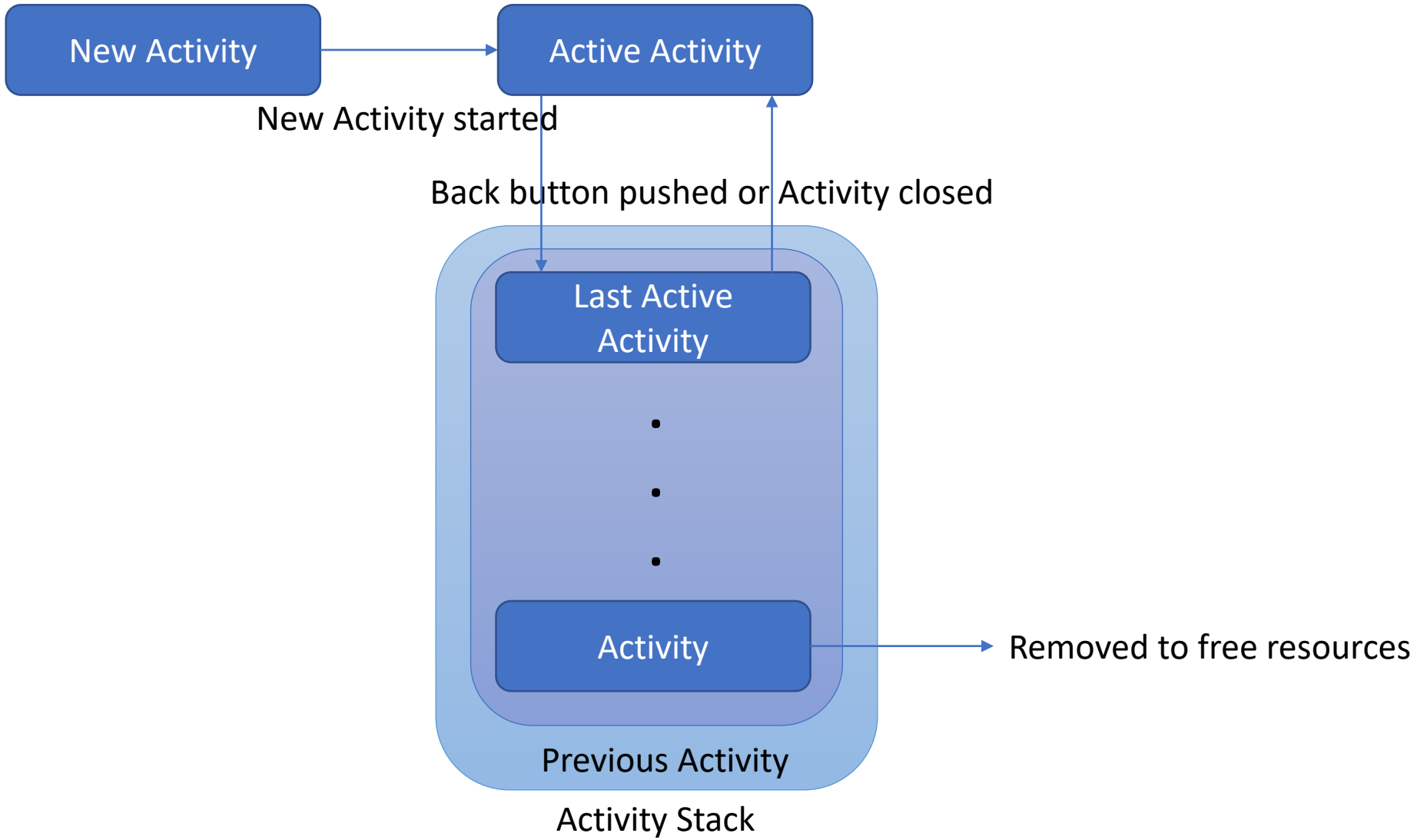
Ciclo di vita di una attività

- Una buona comprensione del ciclo di vita di una Activity è fondamentale per una corretta gestione delle risorse di una applicazione e per garantire la soluzione di continuità nel suo utilizzo all'utente.
- Abbiamo già visto come, una applicazione Android non può controllare il suo ciclo di vita e come il gestore dei Run-Time di Android determina quale applicazione terminare, in caso di mancanza di risorse, sulla base della priorità dell'applicazione stessa; la priorità della applicazione è determinata a sua volta in base allo stato delle attività di cui si compone.

Stato di una Activity

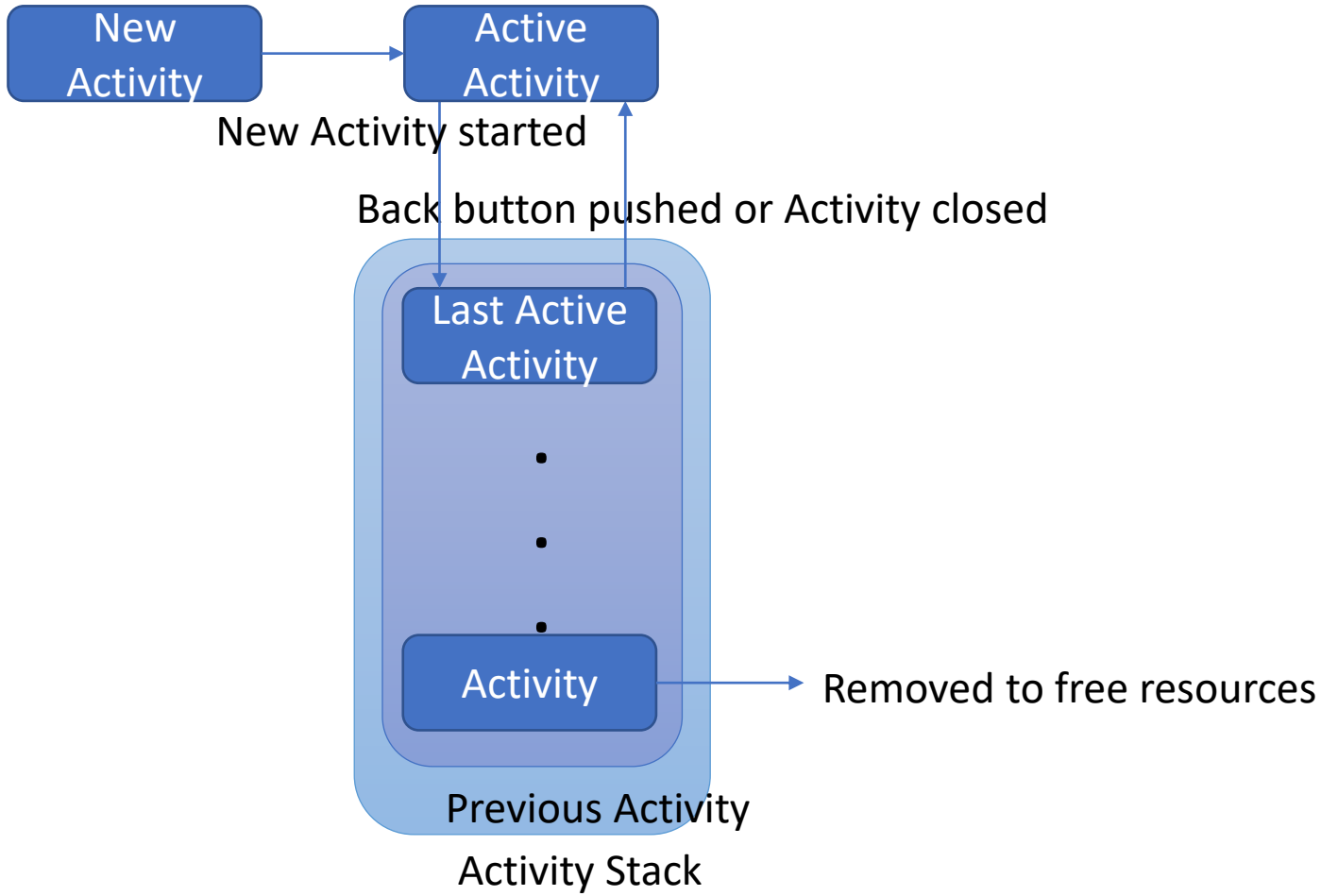
- Lo stato di un Activity è determinato dalla posizione che essa occupa nel cosiddetto Activity Stack: ovvero una pila (LIFO) che colleziona tutte le attività correntemente in esecuzione.
- Quando una nuova Activity inizia e la corrispondente schermata è in foreground, occuperà il top dello Stack.
- Se l'utente aziona il tasto Back o la activity corrente in foreground viene chiusa, la successiva activity dello stack viene mossa nel top e diventa attiva (foreground).

Descrizione processo di una Activity



Descrizione processo di una Activity

- Durante il suo ciclo di vita una activity si muove dentro e fuori lo stack come mostrato nella figura precedente; in ogni istante si potrà trovare in uno di 4 stati possibili...



Stati di una Activity

- **Active**: quando l'Activity è al top dello stack, è visibile, ha il fuoco è in foreground e sta ricevendo l'input dell'utente. Android tenta di tenere in vita questa Activity a tutti i costi, liberando risorse, se necessaria, terminando activity che si trovano sotto nello stack. Quando un'altra Activity diventa Attiva, questa diventerà "Paused".
- **Paused**: in alcuni casi un'Activity può essere visibile, ma non avere il fuoco(transparent o non-full screen Activity attiva); in questo caso è Paused. Quando è in questo stato, l'Activity è trattata come se fosse Active e sarà terminata solo in casi estremi. Se l'Activity viene oscurata completamente passa allo stato "Stopped".
- **Stopped**: l'Activity non è visibile. Anche in questo caso l'Activity rimane in memoria e vengono conservati stato ed informazioni (proprietà, membri). Ovviamente è candidata alla terminazione in caso necessitano risorse. Quando una Activity viene "Stopped" è importante conservare i suoi dati e lo stato corrente della UI. Quando un Activity viene chiusa o essa passa allo stato "Inactive”
- **Inactive**: quando una Activity viene terminata e prima che essa venga lanciata è "Inactive". Tali Activities sono rimossi dallo stack e devono essere fatte ripartire prima che vengano mostrate ed usate.

Android memory manager

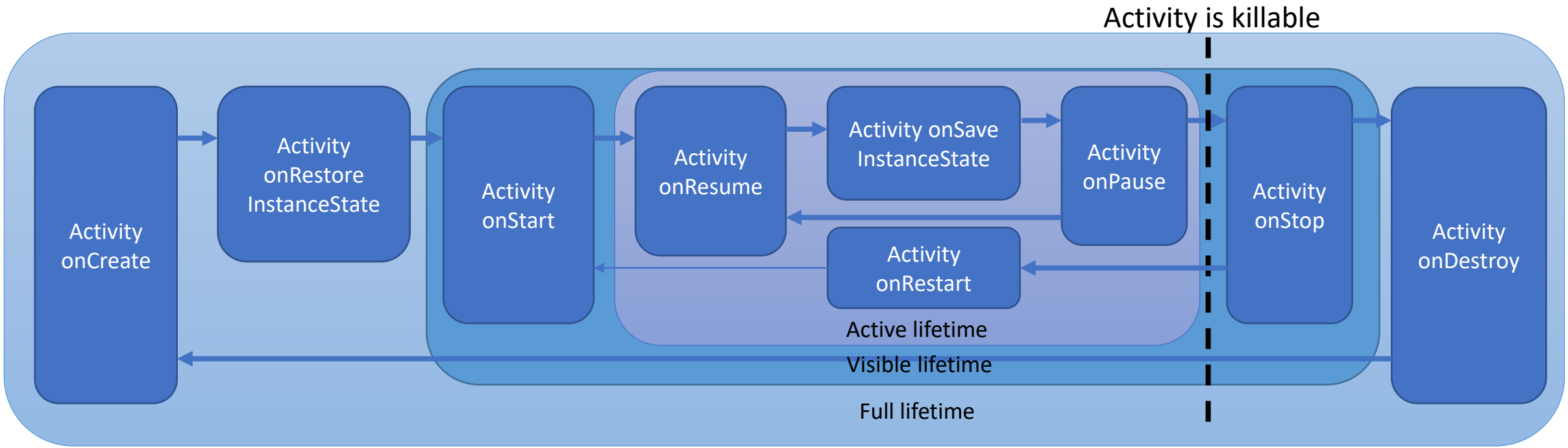
- Il processo di transizione da uno stato all'altro è non-deterministico ed è completamente gestito dall'Android Memory Manager.
- Come è facile intuire Android comincerà a chiudere le applicazioni che contengono Activities nello stato Inactive, prosegue con quelle con Activities Stopped ed in caso estremi con quelle che contengono Activities Paused.
- Come già detto la transizione da uno stato all'altro dovrebbe essere trasparente per l'utente e non dovrebbe fare alcuna differenza lo stato di partenza e/o di arrivo (da paused, stopped,inactive, killed a active).

Gestori eventi Android

- Per assicurare che le attività possano reagire ai cambiamenti di stato, Android fornisce una serie di gestori di eventi che vengono attivati quando un Activity cambia "lifetime" (Ciclo di vita: full, visible e active lifetimes).

Cicli di vita di una Activity

- La figura seguente mostra la corrispondenza tra i tre diversi cicli di vita e gli stati di una activity visti in precedenza; riporta inoltre l'indicazione del nome delle funzioni di callback (event handlers) che dovrebbero essere considerate ad ogni evento corrispondente ad un cambio di stato :



Full Lifetime

- Questo ciclo inizia con la prima chiamata alla funzione onCreate e finisce con l'ultima chiamata alla funzione onDestroy
  - onCreate: si tratta dell'operazione invocata al momento della creazione dell'Activity: utilizzare tale metodo per inizializzare l'attività, costruendo l'interfaccia utente, allocando i riferimenti agli oggetti necessari, fare il binding tra controlli e dati istanziare Servizi e Thread.
    Esso riceve come parametro un oggetto di tipo Bundle, che contiene lo stato della UI salvato nell'ultima chiamata del metodo onSaveInstanceState. Può essere pertanto usato per ripristinare l'UI al suo stato precedente.
  - onDestroy: Ridefinire tale funzione per liberare le risorse create con il metodo onCreate e assicurarsi che tutte le connessioni esterne (rete o database) siano chiuse.

Codice efficiente

- Per avere un codice efficiente è fortemente raccomandato evitare la creazione di oggetti che hanno breve vita in quanto questo comporterebbe operazioni addizionali per il garbage collector che potrebbero influenzare l'efficienza dell'applicazione.

Visible lifetime

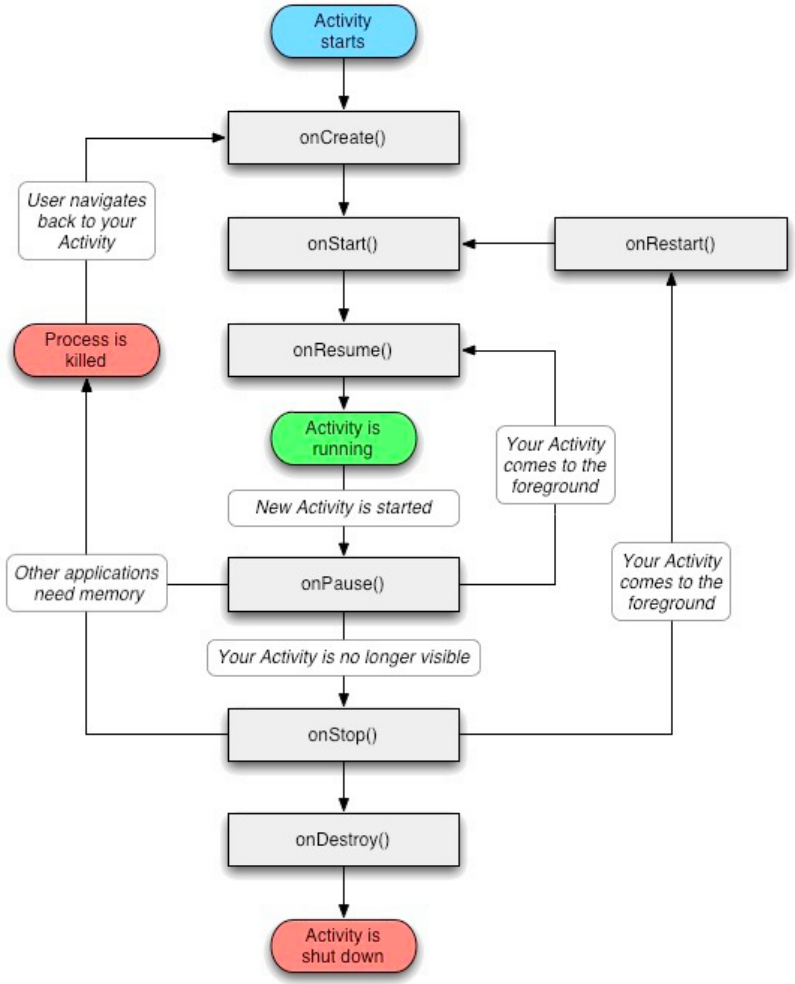
- Tale ciclo inizia con la chiamata onStart e finisce con la chiamata onStop; fra queste chiamate l'Activity sarà visibile all'utente, anche se non ha il focus o è parzialmente oscurata. Durante il suo Full Lifetime passerà diverse volte dal Visible Lifetime: quando ad esempio passa da foreground a background. Solo in casi estremi Android terminerà Activity prima che venga chiamata la funzione onStop.
  - onStop: questo metodo dovrebbe essere usato per mettere in pausa o finire animazione, threads, timer, Services e più in generale attività che sono usate esclusivamente per aggiornare l'interfaccia utente. Utilizzare quindi la funzione onStart oppure onRestart per ripristinare i processi stoppati o in pausa (UI di nuovo visibile).
  - onRestart: questo metodo viene chiamato prima della chiamata a onStart. In questo metodo dovrebbero essere implementate le funzionalità utili per il ritorno nel Full LifeTime
  - onStart/onStop: Utilizzare tali funzioni per registrare e fermare Broadcast Receiver usati solo per l'aggiornamento dell'interfaccia Utente.

Active Lifetime

- Inizia con la chiamata del metodo onResume e finisce con la corrispondente chiamata a onPause.
- Una Activity attiva, è in foreground e riceve l'input dell'utente. Prima di essere distrutta passera in diversi cicli di vita attivi. L'Active Lifetime finirà quando una nuova Activity sarà mostrata, il dispositivo va in pausa, o l'attività perde il focus (viene richiamata la funzione onPause).
- Immediatamente prima della chiamata onPause, viene fatta una chiamata alla funzione di callback onSaveInstanceState. Questo metodo fornisce l'opportunità di salvare lo stato della UI dell'Activity (stato dei bottoni, oggetto con il focus, dati inseriti ma non confermati dell'utente) in un oggetto Bundle che potrà essere passato alle funzioni onCreate o onRestoreInstanceState.

```java
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle icicle) {
    super.onCreate(icicle);

}

// Called after onCreate has finished, use to restore UI state
@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
  super.onSaveInstanceState(savedInstanceState);
  // Restore UI state from the savedInstanceState. This bundle has also
  // been passed to onCreate.
}

// Called before subsequent visible lifetimes for an activity process.
@Override
public void onRestart(){
  super.onRestart();
  // Load changes knowing that the activity has already
  // been visible within this process.
}

// Called at the start of the visible lifetime.
@Override
public void onStart(){
  super.onStart();
  // Apply any required UI change now that the Activity is visible.
}

// Called at the start of the active lifetime.
@Override
public void onResume(){
  super.onResume();
  // Resume any paused UI updates, threads, or processes required
  // by the activity but suspended when it was inactive.
}
```

Flow chart di un Activity

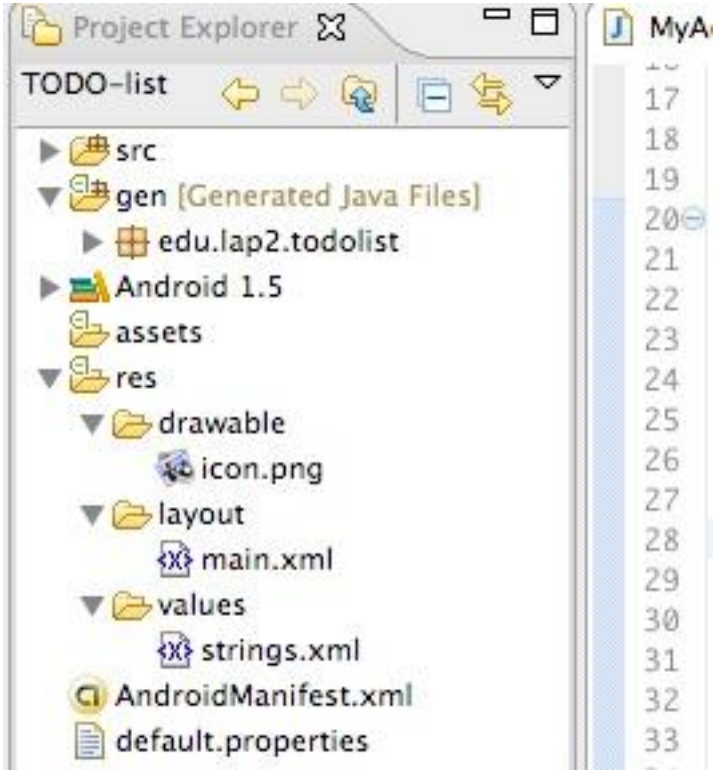


Esternalizzazione delle risorse

- Indipendentemente dall'ambiente di sviluppo che si utilizza è buona norma tenere risorse tipo immagini, costanti etc.. esterne al codice.
- Android supporta l'esternalizzazione delle risorse a cominciare da semplici valori quali stringhe, colori fino a risorse più complesse quali immagini, animazioni e temi e per finire i LAYOUT.

Esternalizzazione delle risorse

- Attraverso il processo di esternalizzazione delle risorse diventa molto più semplice aggiornarle e mantenerle, inoltre è possibile definire diverse risorse per supportare alternativamente hardware e lingue differenti.
- Nell'albero di progetto, le risorse sono memorizzate all'interno della directory res/. Essa conterrà diverse sottocartelle, una per ogni tipologia di risorse.



Tipi di risorse

- Esistono 7 principali categorie di risorse: simple values, drawbles, layouts, animations, XML, styles, raw resources.
- Quando un'applicazione viene compilata, tutte le risorse verranno tradotte (ed ottimizzate per il mobile) e verranno incluse nel package dell'applicazione stessa.
- Il processo suddetto prevede anche la creazione della classe R che per l'appunto contiene i riferimenti ad ogni risorse inclusa nel progetto, e che permette di referenziare la risorsa all'interno del codice (design time syntax checking).

Simple Values

- Simple Values: i valori semplici supportati includono stringhe, colori, dimensioni ed array di stringhe e/o di interi. Tutti i valori sono memorizzati in uno o più file XML nella directory res/values.
- In genere esistono più file (uno per ogni tipologia di valore) ciascuno dei quali all'interno contiene le definizioni, dei valori della tipologia scelta, fatta attraverso i tag opportuni.
  - Strings: le risorse di tipo stringa sono specificate attraverso il tag string come mostrato di seguito:

```
2 <resources>
3     <string name="hello">Hello World, ToDoList!</string>
4     <string name="app_name">TODO-List</string>
5 </resources>
```

Tag e stili di testo

- Android supporta anche semplici stili di testo che possono essere impostati utilizzando i tag html <b>, <i> ed <u>.
- Il tag colors permette di creare nuove risorse di tipo colore. Il valore del colore può essere specificato attraverso il simbolo # seguito opzionalmente dal grado di trasparenza (alpha) e quindi dalla gradazione di rosso, verde e blue espressa in Esadecimale. Possibili notazioni:
  - #RGB
  - #RRGGBB
  - #ARGB
  - #ARRGGBB

```
2 <resources>
3     <color name="blue_opaco">#00F</color>
4     <color name="verde_trasparente">#7700ff00</color>
5 </resources>
```

Dimensions

- Sono molto utili per la creazione di costanti da utilizzare nei layout (spessore bordo, dimensione caratteri etc..). Per definire tali valori si usa il tag ‘dimen’ specificando il valore ed un identificatore che descrive la scala della dimensione secondo la seguente legenda:
  - px: Screen pixels (usa i pixel del display)
  - in: Physical inches (dimensioni assolute espresse in pollici)
  - pt: Physical points (dimensioni espresse in punti 1/72 di pollice)
  - mm: Physical millimeters (dimensioni assolute espresse in millimetri)
  - dp: Density-indipendent pixels relative to a 160 dpi screen (dimensione indipendente dalla risoluzione a 160 dpi vale1 dp = 1 pixel)
  - sp: Scale indipendent pixel (dimensione indipendente dalla risoluzione tiene conto della dimensione impostata per i caratteri).

```
2 <resources>
3   <dimen name="standard_border">5px</dimen>
4   <dimen name="large_font_size">16sp</dimen>
5 </resources>
```

Drawables

- Questa categoria di risorse include immagini bitmaps, NinePatch (PNG stretchable). Tali immagini sono memorizzate come singoli file nella cartella res/drawble; sono supportati formati PNG (raccomandato), GIF e JPG.
  Le immagini NinePatch sono varianti di una immagini PNG in cui viene definita un'area che può essere estesa quando l'immagine stessa viene ingrandita. Per creare una immagine di questo tipo basta disegnare una linea di spessore un pixel attorno all'area che può essere estesa.
- Attenzione: abbiamo già detto che in android le immagini sono risorse di tipo drawable, ma a partire dalla versione 4.3 è stato deciso di creare una nuova tipologia che si chiama per l’appunto mipmap.

Layouts

- Questo tipo di risorsa permette di disaccoppiare il livello di presentazione dal codice grazie al fatto che la progettazione dell'interfaccia utente avviene attraverso file XML. Le risorse Layout quindi permettono di definire l'interfaccia Utente di una Activity : una volta definita essa viene renderizzata in una Activity attraverso la funzione SetContentView (di solito nel metodo onCreate).
- Ogni definzione di Layout risiede in un file XML separato all'interno della cartella res/layout. Il nome del file diventerà un identificatore della risorsa (ricordiamo esempio main.xml e identificatore R.layout.main).

Animations

- Il fatto di poter definire animazioni come risorse esterne permette di riutilizzare la stessa sequenza in diversi e molteplici posti e fornisce l'opportunità di presentare animazioni diverse a seconda dell'hardware e dell'orientazione dello stesso.

Tipi di animazione

- **Tweened Animations**: possono essere usate per ruotare, muovere, strechare e applicare la trasparenza ad una View. Ogni animazione di questo tipo è memorizzata in un file XML nella directory di progetto res/anim. Come per le risorse layout il nome del file fungerà da identificatore della risorsa stessa. Attraverso una animazione si possono modificare i seguenti tag per ciascuno dei quali possono essere impostati i diversi attributi come a fianco mostrato:
- **Frame by Frame Animations**: permette di creare sequenza di risorse drawbles, ognuna delle quali sarà mostrata per un tempo specificato come background di una View. Questo tipo di animazione viene memorizzata nella cartella res/drawble (diversamente dalle Tweened animations) anche in questo caso il nome del file rappresenterà l'identificatore delle risorse.

alpha	fromAlpha/toAlpha	Reale da 0 a 1
scale	fromXScale/toXScale fromYScale/toYScale pivotX/PivotY	Reale da 0 a 1 Reale da 0 a 1 Larghezza/Altezza 0-100%
translate	fromX/toX fromY/toY	Reale da 0 a 1 Reale da 0 a 1
rotate	fromDegrees/toDegrees pivotX/PivotY	Reale da 0 a 360 Larghezza/Altezza 0-100%

```
2  android:oneshot="false">
3  <item android:drawable="@drawable/fbf1" android:duration="500" />
4  <item android:drawable="@drawable/fbf2" android:duration="500" />
5  <item android:drawable="@drawable/fbf3" android:duration="500" />
6 </animation-list>
```

Lista attributi per animazioni

- Ecco una lista di attributi utilizzabili:
  - duration: durata dell'animazione in millisecondi;
  - startOffset: ritardo prima della partenza in millisecondi (se non si utilizza tutte le animazioni di un insieme andranno ad essere eseguite in contemporanea);
  - fillBefore: applicazione delle trasformazioni prima dell'inizio;
  - fillAfter: applicazione delle trasformazioni dopo l'inizio;
  - interpolator: variazione della velocità nel tempo.

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:interpolator="@android:anim/accelerate_interpolator">
  <rotate
    android:fromDegrees="0"
    android:toDegrees="360"
    android:pivotX="50%"
    android:pivotY="50%"
    android:startOffset="500"
    android:duration="1000" />
  <scale
    android:fromXScale="1.0"
    android:toXScale="0.0"
    android:fromYScale="1.0"
    android:toYScale="0.0"
    android:pivotX="50%"
    android:pivotY="50%"
    android:startOffset="500"
    android:duration="500" />
    <alpha
    android:fromAlpha="1.0"
    android:toAlpha="0.0"
    android:startOffset="500"
    android:duration="500" />
</set>
```

Styles

- Risorse di questa categoria consentono alle applicazioni di mantenere un look consistente attraverso la definizione di attributi utilizzati dalle View (memorizzazione di colori e caratteri).
- Lo stile viene impostato attraverso il file manifest.
- Per creare uno stile si usa il tag style che include l'attributo name e contiene uno o più tag item; ogni item dovrebbe includere un attributo name; il tag stesso contiene il valore dell'item specificato attraverso name.
- Il tag Style supporta la ereditarietà attraverso l'attributo parent:

```
<resources>
    <style name="BaseText">
        <item name="android:textSize">14sp</item>
        <item name="android:textColor">#111</item>
    </style>
    <style name="SmallText" parent="BaseText">
        <item name="android:textSize">8sp</item>
    </style>
</resources>
```

Vantaggi nell’uso delle risorse

- Una delle ragioni, e forse la più forte, per esternalizzare le risorse, sta nel meccanismo automatico di selezione delle risorse che offre Android.
- E' possibile creare risorse con valori differenti di configurazione di lingua, location e hardware che Android è in grado di scegliere dinamicamente a run-time.
- Tali risorse vanno specificate utilizzando una apposita struttura di directory "parallela" all'interno della cartella res/ utilizzando una notazione che prevede l'uso del segno "-" (meno - trattino alto) per separare opportunamente identificatori di testo che per l'appunto individuano la caratteristica supportata.

Tipi di qualificatori

- Segue descrizione dei qualificatori utilizzabili:
  - Language: serve a specificare la lingua attraverso le due lettere standard ISO 639-1 (es. en, it)
  - Region: permette di specificare la regione "r" seguito dalle due lettere standard ISO 3166-1-alpha-2 (es. rUs, rGB)
  - Screen-Orientation: usare port (portrait), land (landscape) o square
  - Screen Pixel Density: risoluzione in pixel (es. 92dpi, 108 dpi)
  - Touch Screen Type: notouch, stylus, finger.
  - Keyboard Availability: keysexposed, keyshidden.
  - Keyboard Input Type: nokeys, qwerty, 12key
  - UI Navigation Type: notouch, dpad, trackball
  - Screen Resolution: 320x240

Qualificatori

- E' possibile specificare qualificatori differenti per ogni tipo di risorsa, separandoli tra loro attraverso il "-".
  - Esempi di qualificatori validi:
  - drawable-en-rUS
  - drawable-en-keyshidden
  - drawable-land-notouch-nokeys-320x400
- Esempi di qualificatori non validi:
  - drawable-rUS-en
  - drawable-rUS-rUK