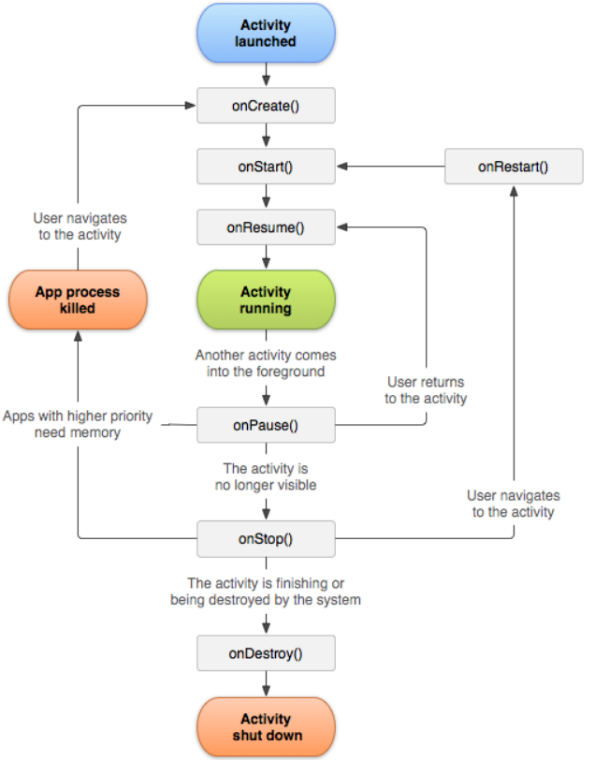
**CICLO DI VITA**

**Il ciclo di vita di un’activity prevede l’esecuzione dei seguenti metodi: onCreate(),** **onStart(), onResume(), onPause(), onStop(), onDestroy(). Indica per ognuno di questi metodi, o almeno per la maggior parte di essi, un’operazione che viene tipicamente implementata nel metodo.**

In onStart(), onResume(), onPause(), onStop(), onDestroy() viene invocato il metodo della superclasse corrispondente senza parametri.

* ***onCreate():*** invocazione del metodo *onCreate()* della superclasse con parametro l’oggetto Bundle e setta visibile il layout che viene descritto nell’activity\_main.
* ***onStart():*** iniziare a disegnare elementi visivi, eseguire animazioni, ecc
* ***onResume():*** provare ad aprire dispositivi ad accesso esclusivo o per accedere a risorse singleton.
* ***onPause():***salvare qualsiasi stato persistente dell'attività in fase di modifica, per assicurarsi che non vada perso nulla se non ci sono risorse sufficienti per avviare la nuova attività senza prima averle ucciso e per fermare le cose che consumano una notevole quantità di CPU per rendere il passaggio all'attività successiva il più velocemente possibile.
* ***onStop():*** per interrompere l'aggiornamento dell'interfaccia utente, l'esecuzione di animazioni e altre cose visive*.*
* ***onDestroy():*** implementato per liberare risorse come thread associati a un'attività, in modo che un'attività distrutta non lasci tali cose mentre il resto dell’applicazione è ancora in esecuzione.

**Si descriva il ciclo di vita di un’activity.**

Ogni ***activity*** di un’applicazione possiede un ciclo di vita. L’activity può trovarsi in sei stati diversi: ***Created***, ***Started***, ***Resumed***, ***Paused***, ***Stopped***, ***Destroyed***. Quando l’app viene creata, il sistema operativo Android crea l’activity principale dell’app: viene invocato il listener ***onCreate()*** e l’activity passa allo stato ***Created***, successivamente, l’activity viene lanciata e messa in ***resume*** (qui, l’utente potrà interagire con l’UI): viene invocato il listener ***onStart()*** e l’activity passa allo stato ***Started***, poi viene invocato il listener ***onResume()*** e l’activity passa allo stato ***Resumed***. Quando l’app lancia una nuova activity, quella corrente viene messa in pausa: si invoca il metodo ***onPause()*** e l’activity passa allo stato ***Paused***; qui, si distinguono tre casi:

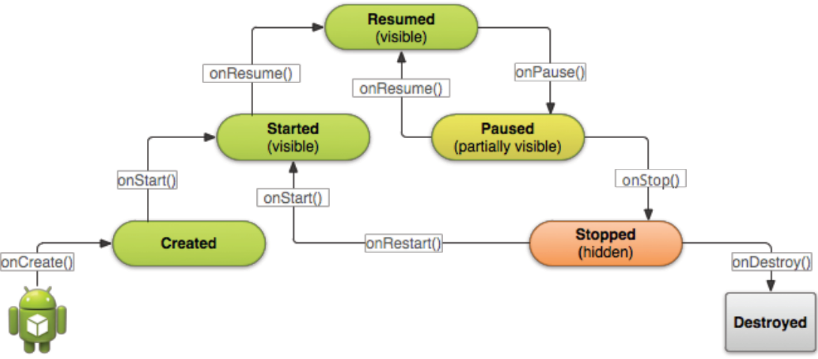
1) Il SO necessita di risorse, per cui distrugge momentaneamente l’activity e ritorna all’***onCreate()***;

2) L’utente ritorna nell’activity, per cui si invoca ***onResume()*** e l’activity passa allo stato ***Resumed***;

3) L’utente chiude l’app, per cui l’activity viene definitivamente distrutta.

Dallo stato ***Paused***, si può passare allo stato ***Stopped*** (prima s’invoca ***onStop()***): qui possono accadere i tre casi precedenti, ma al caso 2 anziché passare allo stato ***Resumed*** l’activity passa allo stato ***Started***, invocando i metodi ***onRestart()*** e ***onStart().*** Infine, l’activity viene distrutta: si invoca il metodo ***onDestroy()*** e l’activity passa allo stato ***Destroyed***.

**Dal momento in cui un’Activity viene lanciata si può trovare in 5 diversi stati. Quali sono questi stati? Quali sono i metodi che permettono di passare dall’uno all’altro? Quali di questi stati sono “transienti” (cioè l’activity rimane per pochissimo tempo in questi stati) e quali invece più “duraturi” (cioè l’activity può rimanere anche per molto tempo in questi stati)?**



***Transienti***: Created, Started, Paused

***Duraturi***: Resumed, Stopped

**L’intero ciclo di vita va dall’esecuzione del metodo onCreate() all’esecuzione del metodo onDestroy(). Da quale a quale metodo vanno invece il periodo di “visibilità” ed il periodo di “foreground” dell’activity?**

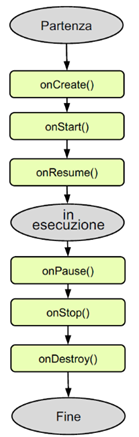
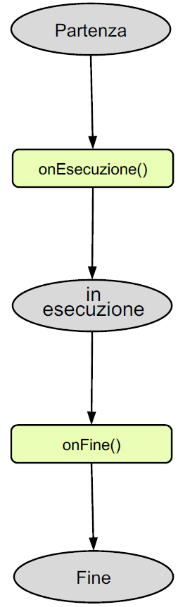
***Visibilità***: *onStart() 🡪 onStop()*

***Foreground***: *onResume()* 🡪 *onPause()*

**Si disegni il ciclo di vita delle attività spiegando quali transizioni avvengono in modo automatico e quali invece per effetto di un intervento o dell’utente o del sistema.**

***Automatico***: onStart(), onResume(), onStop(), onDestroy(), onRestart()

***Intervento*** ***utente/sistema***: onCreate(), onPause()

**II ciclo di vita delle activity, riportato schematicamente a sinistra, prevede l’esecuzione in successione di 3 metodi (onCreate, onStart, onResume) per far partire l’esecuzione di un’app. Perché? Non sarebbe stato meglio avere un solo metodo, come indicato nella figura a destra, nel quale eseguire tutto ciò che viene fatto nei 3 metodi onCreate, onStart, onResume?**

**Analogamente per distruggere un app è prevista l’esecuzione di 3 metodi in successione (onPause, onStop, onDestroy). Non sarebbe stato più semplice avere un solo metodo come indicato nella figura a destra? Motivare la risposta.**

Ogni callback consente di eseguire lavori specifici appropriati per un determinato cambio di stato. Fare il lavoro giusto al momento giusto e gestire correttamente le transizioni rende l’app più solida e performante. Ad esempio, una buona implementazione dei callback del ciclo di vita può aiutare a garantire che l'app eviti:

* Arresto anomalo se l'utente riceve una telefonata o passa a un'altra app durante l'utilizzo dell'app.
* Consumo di preziose risorse di sistema quando l'utente non lo utilizza attivamente.
* Perdere i progressi dell'utente se escono dall'app e ci ritornano in un secondo momento.
* Arresto anomalo o perdita dei progressi dell'utente quando lo schermo ruota tra orientamento orizzontale e verticale.

**Che cosa è il backstack? Supponendo che l’activity A sia l’unica in esecuzione (l’unica presente nel backstack), che la stessa app lancia una nuova activity B che a sua volta lancia l’activity C, che lancia l’activity D, quale è il backstack a questo punto? Cosa succede se dall’activity D si preme il pulsante di “back”? 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?**

Un task è una raccolta di attività con cui gli utenti interagiscono durante l'esecuzione di un determinato lavoro, disposte in uno stack denominato *backstack*. Lo stack ha una struttura LIFO e memorizza le attività nell’ordine della loro apertura. Le attività nello stack non vengono mai riorganizzate. La navigazione del backstack viene eseguita con l’aiuto del pulsante *back*.

La situazione nel backstack è la seguente: D, C, B , A. Se si preme back, ovvero si esegue un pop dallo *stack* dell’activity, l’attività D viene distrutta e C viene messa in foreground.

Se si vuole far in modo che dall’activity D si torni direttamente ad A quando si preme il pulsante di back, occorre lanciare le activity B e C in modo tale che non vengano memorizzate nel backstack: questo dev’essere effettuato programmaticamente, aggiungendo un ***FLAG\_ACTIVITY\_NO\_HISTORY*** all’intent attraverso il metodo ***intent.setFlags(int flag).***

**Si spieghi il meccanismo del backstack. In relazione a tale meccanismo che differenza c’è fra una activity e un frammento?**

Il backstack è un meccanismo che conserva l’ordine di utilizzo delle activity di un’applicazione in una pila:

* all’apertura dell’applicazione viene effettuato un push() della main activity nel backstack, ed un altro push() viene effettuato all’avvio di nuove activity dell’applicazione;
* quando l’utente clicca sul tasto Back (oppure viene cliccato un qualsiasi pulsante che ottiene ciò), viene effettuato un pop() dell’activity corrente.

Le differenze sono:

**Activity** vs **Fragment**

|  |  |
| --- | --- |
| Componente che fornisce un'interfaccia utente in cui l'utente può interagire | Fa parte di un'attività, che contribuisce all’interfaccia utente di tale attività |
| In ***landscape*** possiamo creare solo un riquadro alla volta | In ***landscape*** possiamo creare un multi-riquadro |
| Non dipende dal frammento, ha un proprio ciclo di vita | Dipende dall’attività, non può esistere in maniera indipendente |
| Progetti difficili da gestire | Progetti meglio strutturati e gestione facilitata |
| Può usare 0 o più frammenti | Può essere riutilizzata in altre activity |
| Dobbiamo menzionarlo nel ***Manifest*** | Non è richiesta la dichiarazione nel ***Manifest*** |
| Usa molta memoria | Usa solo la memoria necessaria |
| Processo pesante | Processo leggero |

**A cosa servono i metodi onSaveInstanceState() e onRestoreInstanceState()?**

* ***onSaveInstanceState()*:** viene chiamato prima di posizionare l’activity in stato di background, consentendo di salvare qualsiasi stato di istanza dinamico nella propria attività e nel pacchetto specificato, per essere successivamente ricevuto in onCreate(Bundle) se l’activity deve essere ricreata.
* ***onRestoreInstanceState()***: ulteriore metodo per ripristinare lo stato dell’app, ma non è molto comune, viene chiamato dopo *onStart()*.

**Si consideri il seguente frammento di codice usato per salvare lo stato dell’app prima di onDestroy():**

*public void onSaveInstanceState(Bundle savedInstanceState) {*

*savedInstanceState.putString(”NOME", nome);*

*savedInstanceState.putInt(”NUMERO", numero);*

*super.onSaveInstanceState(savedInstanceState);*

*}*

**Si scriva il corrispondente frammento di codice, indicando anche in quale metodo deve essere inserito, per recuperare lo stato quando l’app viene ricreata.**

Lo si recupera in onCreate():

*@Override*

*protected void onCreate(Bundle savedInstanceState) {*

*…*

*if (savedInstanceState != null) {*

*nome = savedInstanceState.getString ("NOME");*

*numero = savedInstanceState.getInt("NUMERO");*

*}*

*}*

**Si consideri la seguente situazione: un’app viene lanciata e l’utente interagisce con l’app; ad un certo punto l’utente ruota il dispositivo e continua ad interagire con l’app; infine l’utente chiude l’app tramite un apposito pulsante. In quali stati è passata l’app e quali metodi sono stati chiamati dal momento in cui l’app viene lanciata al momento in cui termina?**

1. L’app viene lanciata e l’utente interagisce con l’app:

onCreate() -> Created -> onStart() -> Started -> onResume() -> Resumed;

2. L’utente ruota il dispositivo e continua ad interagire con l’app:

onPause() -> Paused -> onStop() -> Stopped -> onDestroy() -> Destroyed -> [punto 1]

3. L’utente chiude l’app tramite un apposito pulsante:

onPause() -> Paused -> onStop() -> Stopped -> onDestroy() -> Destroyed.

**Si spieghi come un’app possa eseguire la seguente sequenza di metodi**

1. **onCreate()**
2. **onStart()**
3. **onResume()**
4. **onPause()**
5. **onStop()**
6. **onRestart()**
7. **onStart()**
8. **onResume()**
9. **onPause()**
10. **onResume()**

L’utente avvia l’app ed interagisce con essa.

L’utente passa ad un'altra app, per poi riprenderla successivamente.

L’utente chiude l’app per poco tempo, e la riprende.

**Un dispositivo Android può funzionare sia in modalità portrait (verticale) che landscape (orizzontale). Quando un dispositivo Android viene ruotato si passa dall’una all’altra modalità. Per gestire in maniera appropriata tali passaggi, di cosa si deve preoccupare il programmatore?**

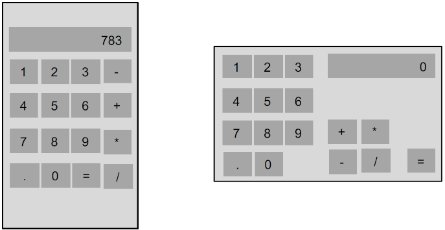
Quando l’utente ruota il dispositivo, l’activity viene prima eliminataeseguendo le callback: *onPause()* 🡪 *onStop()* 🡪 *onDestroy()*, e poi ricreata: *onCreate()* 🡪 *onStart()* 🡪 *onResume().* Dunque si ha una perdita di stato.

Quando un dispositivo Android viene ruotato, passando da una modalità (portrait/landscape) all’altra (landscape/portrait), normalmente un’applicazione effettua un “cambio di configurazione”, pertanto necessita di esser prima distrutta per poi esser ricreata: questo fa sì che si perda lo stato dell’activity che l’utente utilizzava, se il programmatore non gestisce il cambiamento di stato.

Per gestire tale cambiamento, di recente è stato implementato un attributo di <application>, android:configChanges=”orientation | keyboardHidden | screenSize”: se lo si specifica, è possibile lasciare al sistema operativo la gestione, così che non venga perso alcun dato; tale attributo fa sì che venga invocato il listener onConfigurationChanged(), in cui è possibile eseguire qualsiasi computazione si desideri.

Un approccio programmatico che consente al programmatore la gestione manuale dei dati consiste nel salvare alcuni dati nell’oggetto Bundle savedInstanceState: quando c’è il cambiamento viene invocato il metodo onSaveInstanceState(Bundle savedInstanceState), al cui interno vengono tipicamente aggiunte informazioni a tale oggetto (usando i setter savedInstanceState.putStringArrayList(String tag, ArrayList<String> list), savedInstanceState.putInt(String tag, int value), …) e, prima della chiusura del metodo, viene invocato super.onSaveInstanceState(savedInstanceState) - metodo di Activity - per rendere effettivo il salvataggio.

Questo oggetto viene passato anche al metodo onCreate(), per cui qui è possibile ricavare le informazioni sullo stato memorizzate attraverso i getter savedInstanceState.getStringArrayList(String tag), savedInstanceState.getInt(String tag), …

**Stai usando il tuo smartphone per fare dei conti con un app “calcolatrice”, come mostrato nella figura a sinistra. Involontariamente ruoti il dispositivo e ti ritrovi con la situazione descritta nella figura a destra. Il valore “783” è diventato “0” e la disposizione dei tasti è cambiata. Cosa è successo? Cosa ha fatto bene e cosa ha sbagliato il programmatore dell’app? Cosa avrebbe dovuto fare per ovviare all’errore?**

Quando si ruota il dispositivo, l’app con cui l’utente sta interagendo, se non è stato ben gestito il cambio di configurazione, viene prima distrutta e poi riavviata (Quesito 3, punto 2), per cui (non essendo stato gestito il cambio) viene perso lo stato precedente.

La disposizione dei tasti è cambiata, giustamente, in quanto si è passati da portrait a landscape, per cui è necessario ridisegnare il layout: in questo caso, il programmatore ha gestito elegantemente il tutto.

Per non perdere lo stato precedente, di recente è stato implementato l’attributo android:configChanges=”orientation | screenSize | keyboardHidden”, che consente all’app di non perdere lo stato precedente quando cambia l’orientamento del dispositivo, e permette di eseguire una qualsiasi computazione desiderata invocando il listener onConfigurationChanged().

Un altro modo per tener traccia del valore calcolato (es. “783”) è utilizzare il listener onSaveInstanceState(Bundle savedInstanceState), invocato prima dell’onDestroy(), che consente di aggiungere all’oggetto Bundle vari dati attraverso metodi putInt(String tag, int value), putFloat(String tag, float value), …; infine, per conservare i cambiamenti occorre invocare super.onSaveInstanceState(savedInstanceState). Per riottenere i vari dati memorizzati nel Bundle, è possibile utilizzare l’oggetto Bundle passato all’onCreate(), invocandovi getInt(String tag), getFloat(String tag), … ed inserire il valore nel TextView del risultato.

Così facendo, viene memorizzato lo stato precedente.

**L’app TreSchermi consta di 3 activity, Activity1, Activity2 e Activity3. Ognuna di queste activity prevede 3 pulsanti: uno per passare all’attività successiva (Next), uno per passare all’attività precedente (Prev) ed una per terminare l’app (Stop). Il pulsante Next dell’Activity3 lancia l’Activity1 ed il pulsante Prev dell’Activity1 lancia l’Activity3. Si indichi la sequenza delle chiamate delle funzioni del ciclo di vita delle 3 attività nei seguenti due casi:**

**1. Avvio app (Activity1) – Pulsante Next – Pulsante Next – Pulsante Next – Pulsante Stop**

**2. Avvio app (Activity1) – Pulsante Prev – Pulsante Next – Pulsante Stop**

**CASO 1 CASO 2**

|  |  |
| --- | --- |
| 1. Activity1.onCreate() 2. Activity1.onStart() 3. Activity1.onResume() 4. Activity1.onPause() 5. Activity2.onCreate() 6. Activity2.onStart() 7. Activity2.onResume() 8. Activity2.onPause() 9. Activity3.onCreate() 10. Activity3.onStart() 11. Activity3.onResume() 12. Activity3.onPause() 13. Activity1.onResume() 14. Activity1.onPause() 15. Activity1.onStop() 16. Activity1.onDestroy() 17. Activity2.onStop() 18. Activity2.onDestroy() 19. Activity3.onStop() 20. Activity3.onDestroy() | 1. Activity1.onCreate() 2. Activity1.onStart() 3. Activity1.onResume() 4. Activity1.onPause() 5. Activity3.onCreate() 6. Activity3.onStart() 7. Activity3.onResume() 8. Activity3.onPause() 9. Activity1.onResume() 10. Activity1.onPause() 11. Activity1.onStop() 12. Activity1.onDestroy() 13. Activity3.onStop() 14. Activity3.onDestroy() |

**Nel contesto dell’esercizio precedente, si mostri un frammento di codice che potrebbe essere per l’onClickListener del pulsante Next dell’Activity2.**

|  |
| --- |
| *next.setOnClickListener(new View.OnClickListener() {*  *@Override*  *public void onClick(View v) {*  *Intent i = new Intent(this, Activity3.class);*  *startActivity(i);*  *}*  *});* |