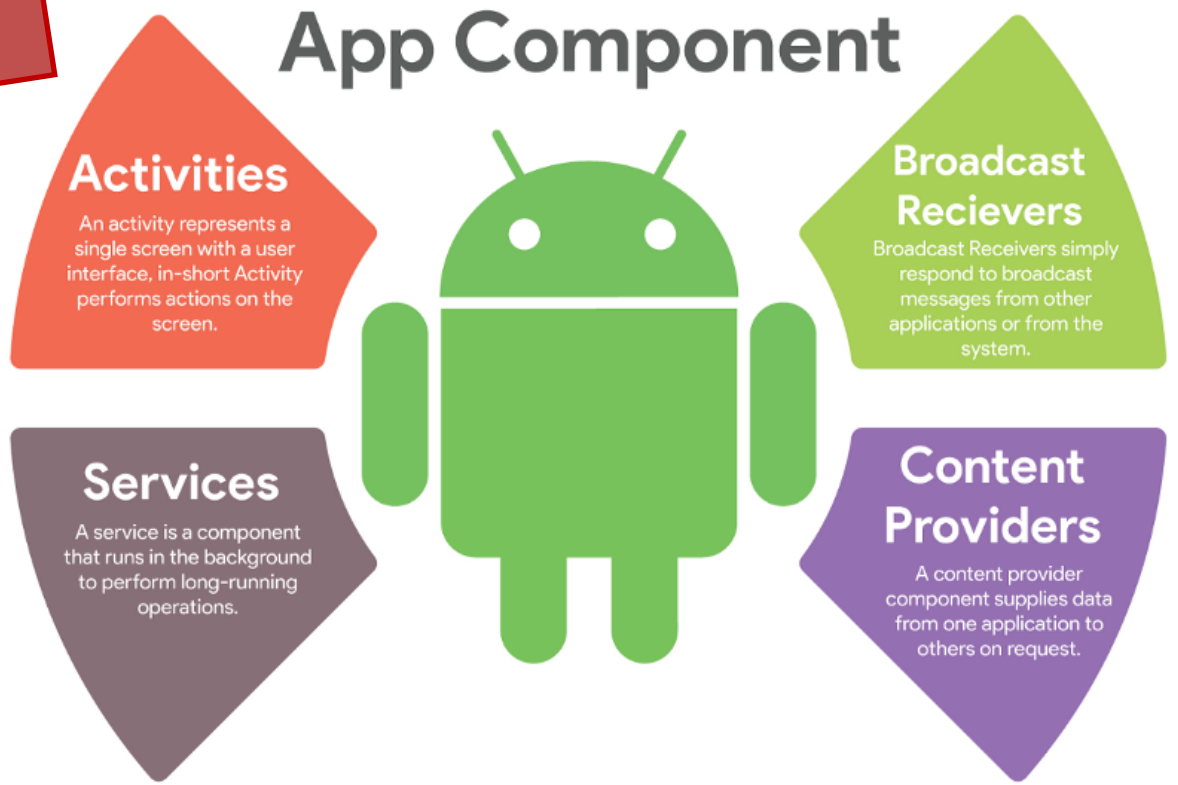
Android UI - Activity

Lezione 5

Activity

- Sappiamo che:
  - È uno dei quattro compenti fondamentali in Android
  - Rappresenta un entry point per interagire con un'app
  - È l'unico componente con un interfaccia
  - Un'app può avere diverse activity (e quindi diverse interfacce)



Activity

- Nel progettare un'app e, di conseguenza, le sue interfacce, devi pensare che un'app mobile è diversa da un'applicazione desktop
  - L'interazione non è pre-determinabile
    - Pensa per esempio ad un'app di email, quando la apri vedi un'interfaccia, ma se invece da un'altra app richiedi di scrivere un'email, si aprirà direttamente la schermata di scrittura – il flusso è stato iniziato da un'altra app
    - Ciò è possibile perché l'app non è un unico elemento atomico, ma è composto da diverse Activity
- La classe Activity è progettata per facilitare questo paradigma!
- Un activity è implementata come una sottoclasse della classe Activity

Activity: altri dettagli

- Un’activity fornisce la finestra in cui l'app «disegna» la sua interfaccia utente
- Questa finestra in genere riempie lo schermo, ma può essere più piccola dello schermo e fluttuare sopra altre finestre
- In genere, un’activity implementa una schermata in un’app
  - Ad esempio, una delle activity di un'app può implementare una schermata Preferenze, mentre un'altra attività implementa una schermata Seleziona foto, ecc…

Activity: altri dettagli

- La maggior parte delle app contiene diverse schermate e quindi più activity
- In genere, un'activity in un'app viene specificata come **main activity**, ovvero è la prima schermata visualizzata quando l'utente avvia l'app
- Ogni activity può quindi avviare un'altra activity per eseguire azioni diverse
  - Ad esempio, la main activity in una semplice app di posta elettronica può fornire la schermata che mostra una casella di posta elettronica. Da lì, la main activity potrebbe avviare altre activity che forniscono schermate per azioni come la scrittura di e-mail e l'apertura di singole e-mail

Activity e manifest

- Sebbene le activity «lavorino» insieme per formare un'esperienza utente coerente in un'app, ogni activity è solo vagamente legata alle altre activity
  - di solito ci sono dipendenze minime tra le activity in un’app
- Inoltre, le activity spesso avviano activity appartenenti ad altre app
  - Ad esempio, un'app browser potrebbe avviare l’activity «Condividi» di un'app di social media
- Per utilizzare le activity nella tua app, devi registrare le relative informazioni nel **manifest** dell'app e devi gestire i cicli di vita delle activity in modo appropriato

File AndroidManifest.xml

- Ogni app project deve avere un file AndroidManifest.xml (con esattamente quel nome) nella radice del progetto
- Il file manifest descrive le informazioni essenziali sulla tua app per gli strumenti di compilazione di Android, il sistema operativo Android e Google Play

Activity.kt | AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyApplication">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="My Application"
            android:theme="@style/Theme.MyApplication.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

File AndroidManifest.xml

- Tra le altre cose, il file manifest deve dichiarare
  - I componenti dell'app, che includono tutte le **activity**, i services, broadcast receiver e content provider. Ogni componente deve definire proprietà di base come il nome della sua classe. Il componente può anche dichiarare funzionalità come quali configurazioni del dispositivo può gestire e intent filter che descrivono come avviare il componente
  - Le autorizzazioni necessarie all'app per accedere a parti protette del sistema o ad altre app. Dichiara inoltre eventuali autorizzazioni che altre app devono avere se desiderano accedere ai contenuti da questa app
  - Le funzionalità hardware e software richieste dall'app, che influiscono sui dispositivi che possono installare l'app da Google Play
- Utilizzando Android Studio, il file manifest viene creato automaticamente e la maggior parte degli elementi manifest essenziali vengono aggiunti durante la creazione dell'app

Activity e manifest

- **Importante**: Affinché la tua app sia in grado di utilizzare le activity, devi dichiararle insieme ai relativi attributi nel file manifest
- Eventualmente, sempre nel manifest devi anche dichiarare
  - Intent-filter
  - permission

Dichiarazione nel manifest: <activity>

Per dichiarare la tua attività, apri il file manifest e aggiungi un elemento <activity> come figlio dell'elemento <application> Per esempio:

Il nome è l'unico attributo richiesto! Se ne possono aggiungere altri opzionali quali label, icon, o UI theme

```
<manifest ... >
  <application ... >
      <activity android:name=".ExampleActivity" />
      ...
  </application ... >
  ...
</manifest >
```

Un Intent Filter in Android è un elemento dichiarativo utilizzato nelle applicazioni per specificare i tipi di intenti (Intent) che un componente può gestire. È definito nel file AndroidManifest.xml e permette di rendere un'Activity, un Service o un Broadcast Receiver accessibile per determinati eventi o azioni.

Activity e Intent filter

- Intent filter permettono di eseguire un'activity sfruttando richieste sia *explicit* che *implicit*
  - *Esempio:*
    - Una richiesta esplicita potrebbe dire al sistema "Avvia l'attività Send e-mail nell'app Gmail". Al contrario, una richiesta implicita dice al sistema di "Avviare una schermata Send e-mail in qualsiasi activity che può svolgere il lavoro".
  - Quando l'interfaccia utente del sistema chiede ad un utente quale app utilizzare durante l'esecuzione di un'attività, si tratta di un Intent filter
  - Questi filter si dichiarano nel manifest, all'interno dell'elemento Activity

Activity e Intent filter: esempio manifest

```
<activity android:name=".ExampleActivity"
android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
</activity>
```

Dichiarando <*category*> a «default» si consente all'activity di **ricevere richieste di avvio**;
<*data*> permette di specificare che tipo di dati questa activity può inviare

Activity e Intent filter

- Mentre, in kotlin per chiamare l'activity occorre fare:

```
val sendIntent = Intent().apply {
    action = Intent.ACTION_SEND
    type = "text/plain"
    putExtra(Intent.EXTRA_TEXT, textMessage)
}
startActivity(sendIntent)
```

Activity e Intent filter: esempio manifest

- Nel esempio appena visto, dichiariamo quindi che l'app può **ricevere richieste di invio di dati di tipo testuale**
- Se vuoi che la tua app sia autonoma e non consenta ad altre app di attivare le sue activity, non hai bisogno di intent filter
  - Le activity che non vuoi rendere disponibili ad altre applicazioni non dovrebbero avere intent filter e puoi avviarle direttamente utilizzando intent espliciti

Permission e manifest

- Puoi utilizzare il tag <activity> del manifest per controllare quali app possono avviare una determinata activity
- Un’activity non può avviare un’altra activity a meno che entrambe le attività non dispongano delle stesse autorizzazioni nel manifest

Permission e manifest

- Ad esempio, se la tua app desidera utilizzare un'ipotetica app denominata SocialApp per condividere un post sui social media,
- SocialApp stessa deve definire l'autorizzazione che deve avere un'app che la chiama:

```
<manifest>
<activity android:name="...."
  android:permission=”com.google.socialapp.permission.SHARE_POST”

/>
```

E, la tua app:

```
<manifest>
  <uses-permission android:name="com.google.socialapp.permission.SHARE_POST" />
</manifest>
```

Permission e manifest

- Ad esempio, se la tua app desidera utilizzare un'ipotetica app denominata SocialApp per condividere un post sui social media,
- SocialApp stessa deve definire l'autorizzazione che deve avere un'app che la chiama:

```
<manifest>
<activity android:name="...
```

I permessi lì vedremo meglio più avanti!

E, la tua app:

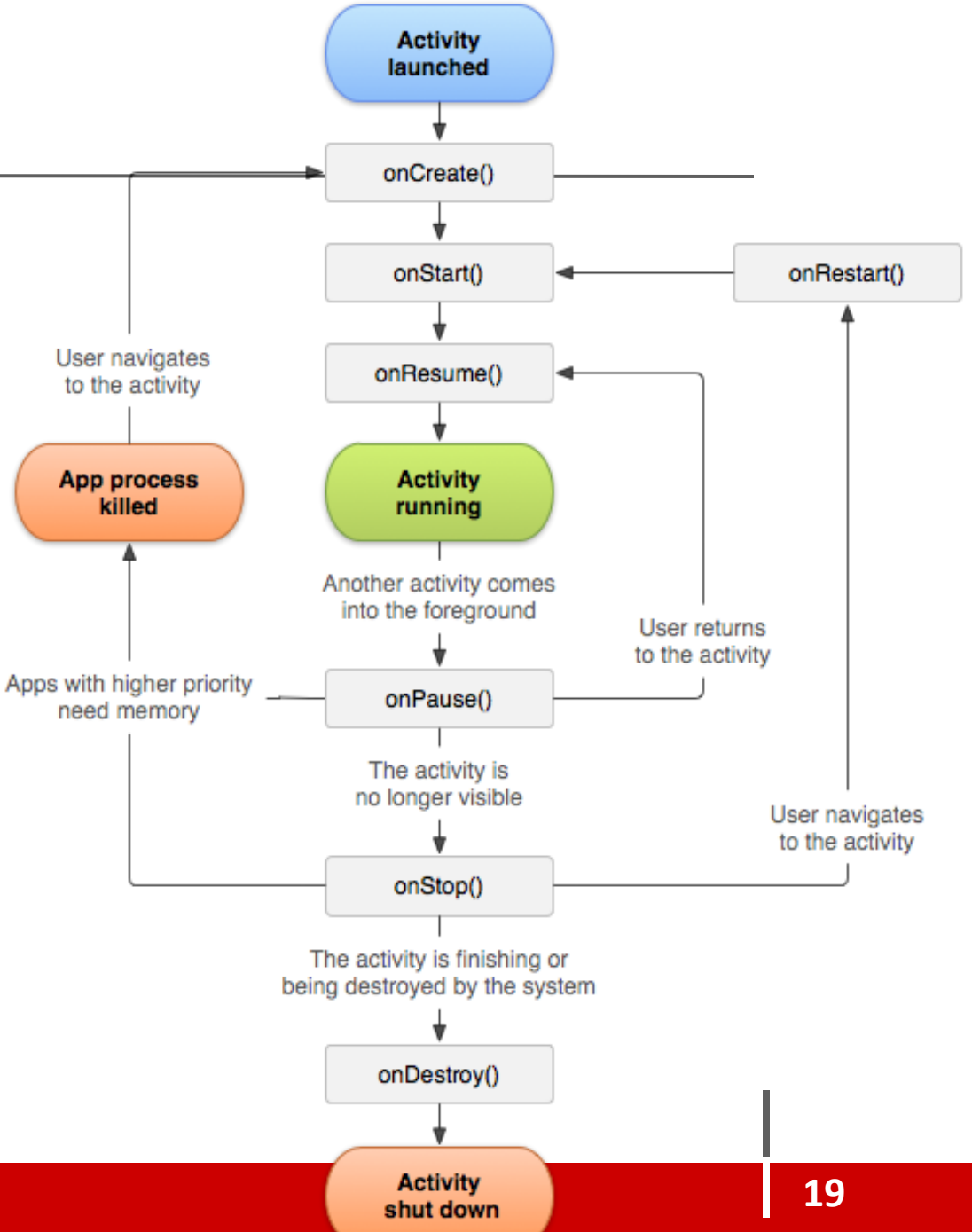
```
<manifest>
  <uses-permission android:name="com.google.socialapp.permission.SHARE_POST" />
</manifest>
```

Activity e ciclo di vita

- Nel corso della sua vita, un'activity passa attraverso una serie di stati
- Si utilizza una serie di callback per gestire le transizioni tra gli stati
  - In altre parole, mentre un utente naviga attraverso l'app, apre e chiude e poi torna nell'app, le istanze di activity nella tua app passano attraverso diversi stati nel loro ciclo di vita
- Gestire bene gli stati, fare la cosa «giusta» al momento giusto e gestire correttamente le transizioni rende la tua app più robusta e performante, e aumenta l'UX dell'utente

Ciclo di vita di un activity

- La classe Activity fornisce una serie di callback che consentono all'attività di **sapere che uno stato è cambiato**: se il sistema sta creando, arrestando o riprendendo un'attività o sta distruggendo il processo in cui risiede l'attività
- All'interno dei metodi di callback del ciclo di vita, puoi dichiarare come si comporta la tua activity in ogni stato
- NOTA: non occorre per forza gestire tutti gli stati! Dipende dallo scopo dell'app!

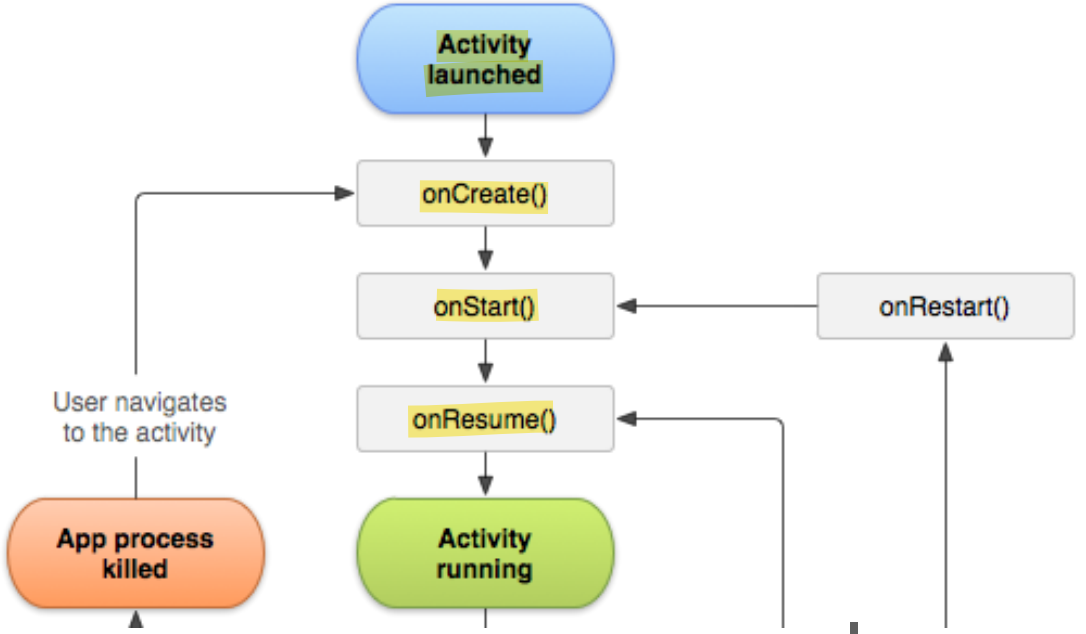


Ciclo di vita di un activity

- È molto importante capire come funziona il ciclo di vita di un activity così da evitare
  - Che l'app smetta di funzionare inaspettatamente quando avviene un evento che interrompe il normale flusso di utilizzo dell'app
    - Per esempio: l'utente riceve una notifica o risponde al telefono
  - Che l'app consumi risorse non necessarie
  - Di perdere i progressi dell'utente se lascia l'app e vi torna in un secondo momento
  - Per evitare l'arresto anomalo o la perdita dei progressi dell'utente quando lo schermo ruota tra orientamento orizzontale e verticale.
  - Per far si che l'utente possa stoppare l'app senza alcun rischio e possa tornare ad utilizzarla in un altro momento (senza aver corrotto l'applicazione stessa o i dati da essa utilizzati)

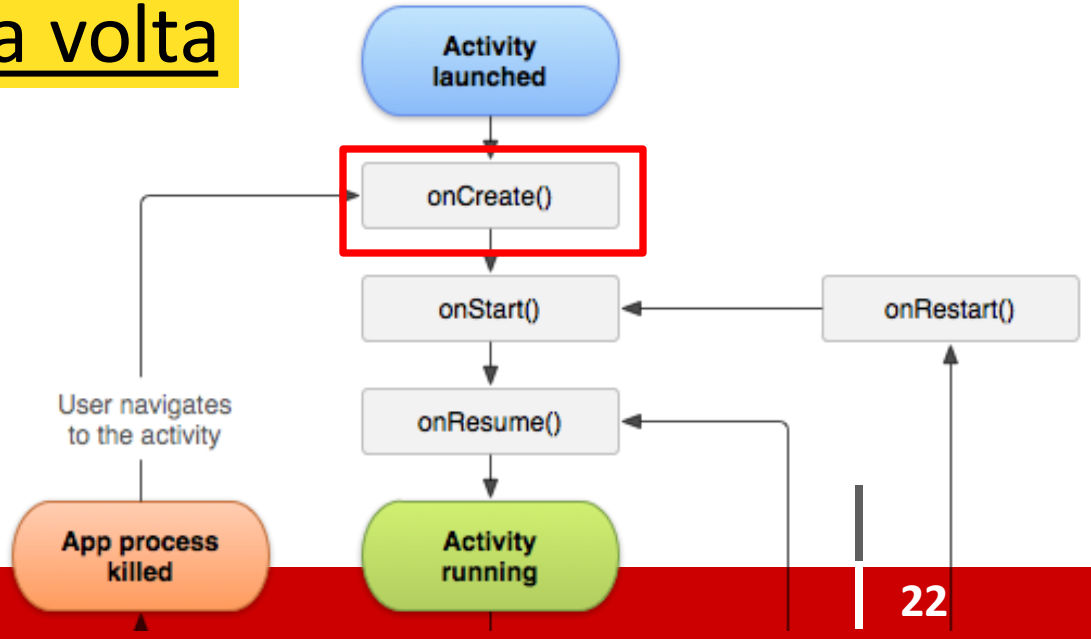
Activity in esecuzione

- Quando un'activity va in esecuzione per interagire direttamente con l'utente vengono obbligatoriamente invocati tre metodi:
  - onCreate()
  - onStart()
  - onResume()



onCreate()

- L’activity viene creata. Il programmatore deve assegnare le configurazioni di base e definire quale sarà il layout dell’interfaccia
- In questo stato si esegue una logica di avvio dell'applicazione di base che dovrebbe avvenire una sola volta per l'intera durata dell’activity
- Al termine di onCreate(), il callback successivo è sempre onStart()



Nel codice di default di una empty activity

```
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Text

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Text("Hello world!")
        }
    }
}
```

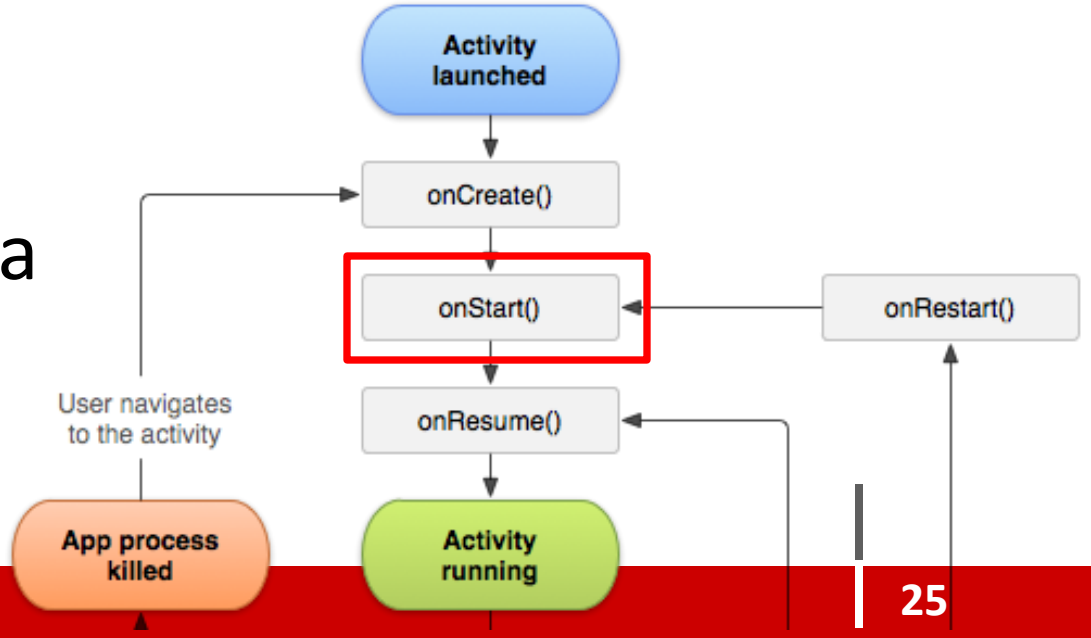
Creata con Compose!

onCreate() e *savedInstanceState*

- Questo metodo riceve il parametro *savedInstanceState*, che è un oggetto Bundle contenente lo stato precedentemente salvato dell'attività
- Se l'attività non è mai esistita prima, il valore dell'oggetto Bundle è null

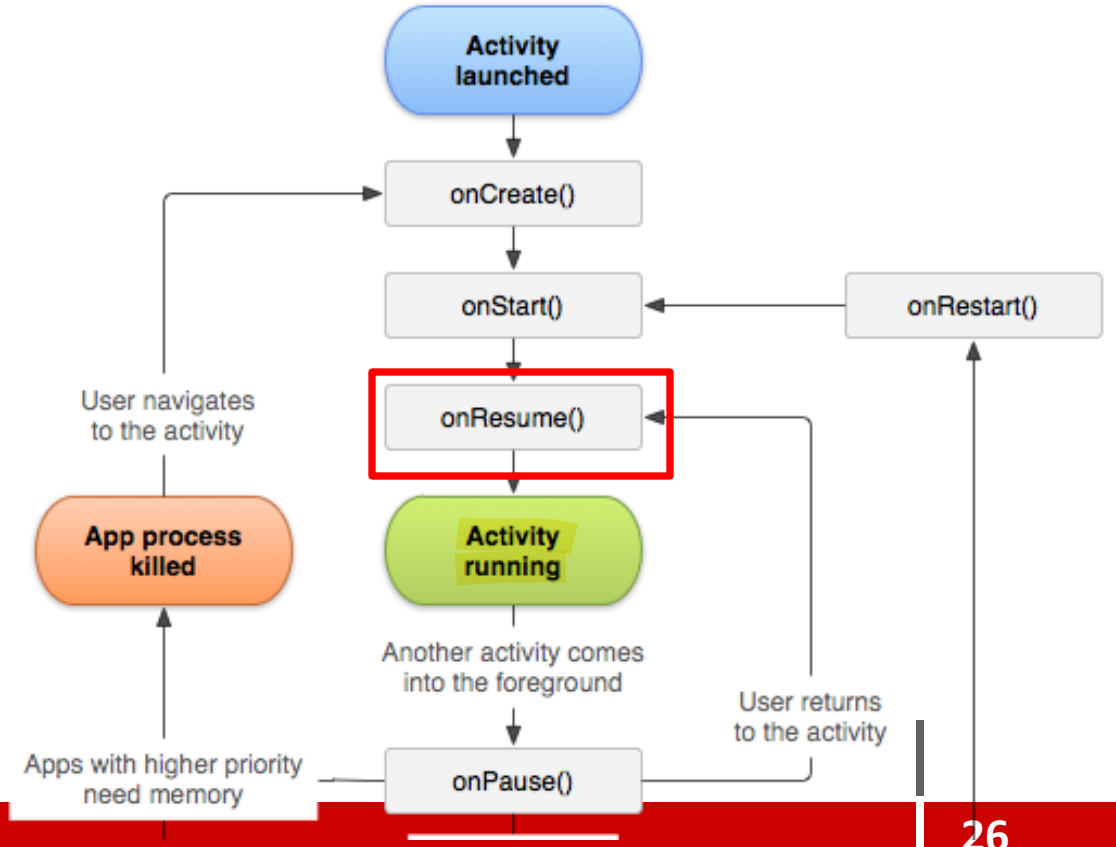
onStart()

- L'activity diventa visibile
  - È il momento in cui si possono attivare funzionalità e servizi che devono offrire informazioni all'utente
- Questo callback contiene ciò che equivale ai preparativi finali dell'activity per essere in primo piano e diventare interattiva
- Appena finita questa fase, l'app entra nello stato..



onResume()

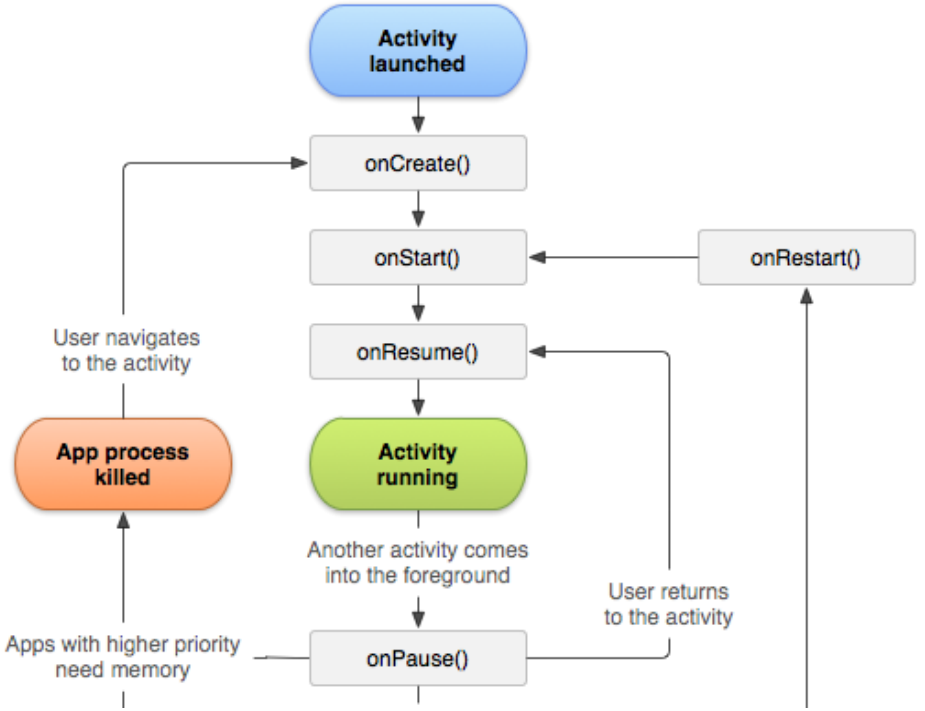
- L’activity diventa la destinataria di tutti gli input dell’utente
- Il sistema richiama questo callback appena prima che l'attività inizi a gestire le interazioni con l'utente
- A questo punto, l’activity si trova nella parte superiore dello stack e acquisisce tutti gli input dell'utente
- La maggior parte delle funzionalità principali di un'app è implementata nel metodo onResume()



onResume()

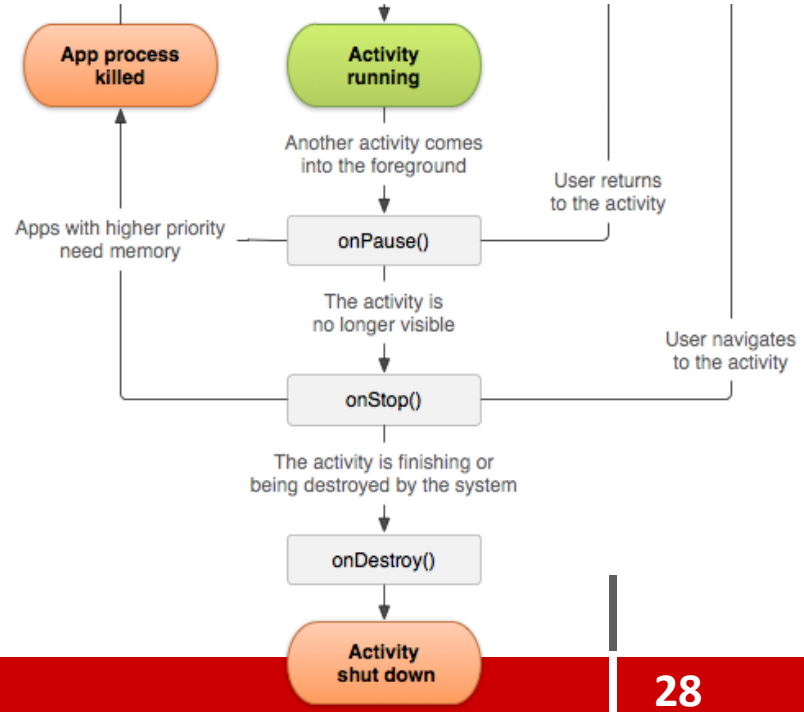
- L’app rimane in questo stato finché non succede qualcosa che distoglie l'attenzione dall’app
  - Come, esempio, la ricezione di una telefonata, la navigazione dell'utente verso un'altra attività o lo spegnimento dello schermo del dispositivo, ecc.

- Quando si verifica un evento di interruzione, l'attività entra nello stato Paused e il sistema richiama il callback onPause()
  - Il callback onPause() segue sempre onResume ()

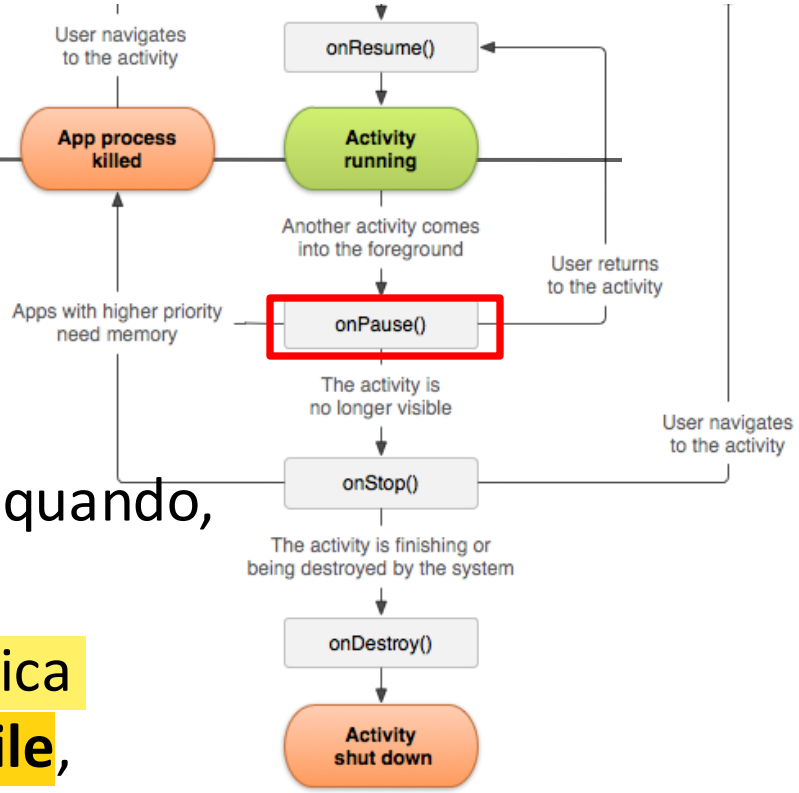


Activity in «background»

- Android pone a riposo l'activity nel momento in cui l'utente sposta la sua attenzione su un'altra attività del sistema, ad esempio apre un'applicazione diversa, riceve una telefonata o semplicemente – anche nell'ambito della stessa applicazione – viene attivata un'altra Activity
- Anche questo percorso, passa per tre metodi di callback:
  - **onPause** (contrapposto di onResume)
  - **onStop** (contrapposto di onStart)
  - **onDestroy** (contrapposto a onCreate)



onPause()



- Il sistema chiama onPause() quando l'attività perde lo stato attivo ed entra in uno stato di pausa. Questo stato si verifica quando, ad esempio, l'utente tocca il pulsante «Indietro» o «Recenti»
- Quando il sistema chiama onPause() per la tua activity, significa tecnicamente che la tua activity è ancora **parzialmente visibile**, ma il più delle volte indica che l'utente sta lasciando l'activity e l'activity entrerà presto nello stato Stopped o Resumed
- Una volta che onPause() termina l'esecuzione, il callback successivo è onStop() o onResume(), a seconda di cosa succede dopo che l'activity entra nello stato onPause()

onPause()

- Esistono diversi motivi per cui un'attività può entrare in questo stato
- Per esempio:
  - Alcuni eventi interrompono l'esecuzione dell'app, come detto prima quando parlavamo di onResume()
  - In Android 7.0 (livello API 24) o versioni successive, più app vengono eseguite in modalità multifinestra. Poiché solo una delle app (finestre) ha lo stato attivo in qualsiasi momento, il sistema mette in pausa tutte le altre app
  - Si apre una nuova activity semitrasparente (come una finestra di dialogo). Finché l'activity è ancora parzialmente visibile ma non a fuoco, rimane in pausa

onPause()

- Puoi utilizzare il metodo onPause() per rilasciare risorse di sistema, handle ai sensori (come il GPS) o qualsiasi risorsa che possa influire sulla durata della batteria mentre l'attività è in pausa e l'utente non ne ha bisogno
- Tuttavia, come menzionato nella slide precedente, un'activity in pausa potrebbe essere ancora completamente visibile se in modalità multi-finestra
  - Pertanto, dovresti prendere in considerazione l'utilizzo di onStop() invece di onPause() per rilasciare completamente o regolare le risorse e le operazioni relative all'interfaccia utente per supportare meglio la modalità multi-finestra

onPause()

- L'esecuzione di onPause() è molto breve e non offre necessariamente tempo sufficiente per eseguire operazioni di salvataggio
- Per questo motivo, NON dovresti usare onPause() per salvare i dati dell'applicazione o dell'utente, effettuare chiamate di rete o eseguire transazioni di database; tale lavoro potrebbe non essere completato prima del completamento del metodo
  - Ancora una volta, meglio usare onStop()

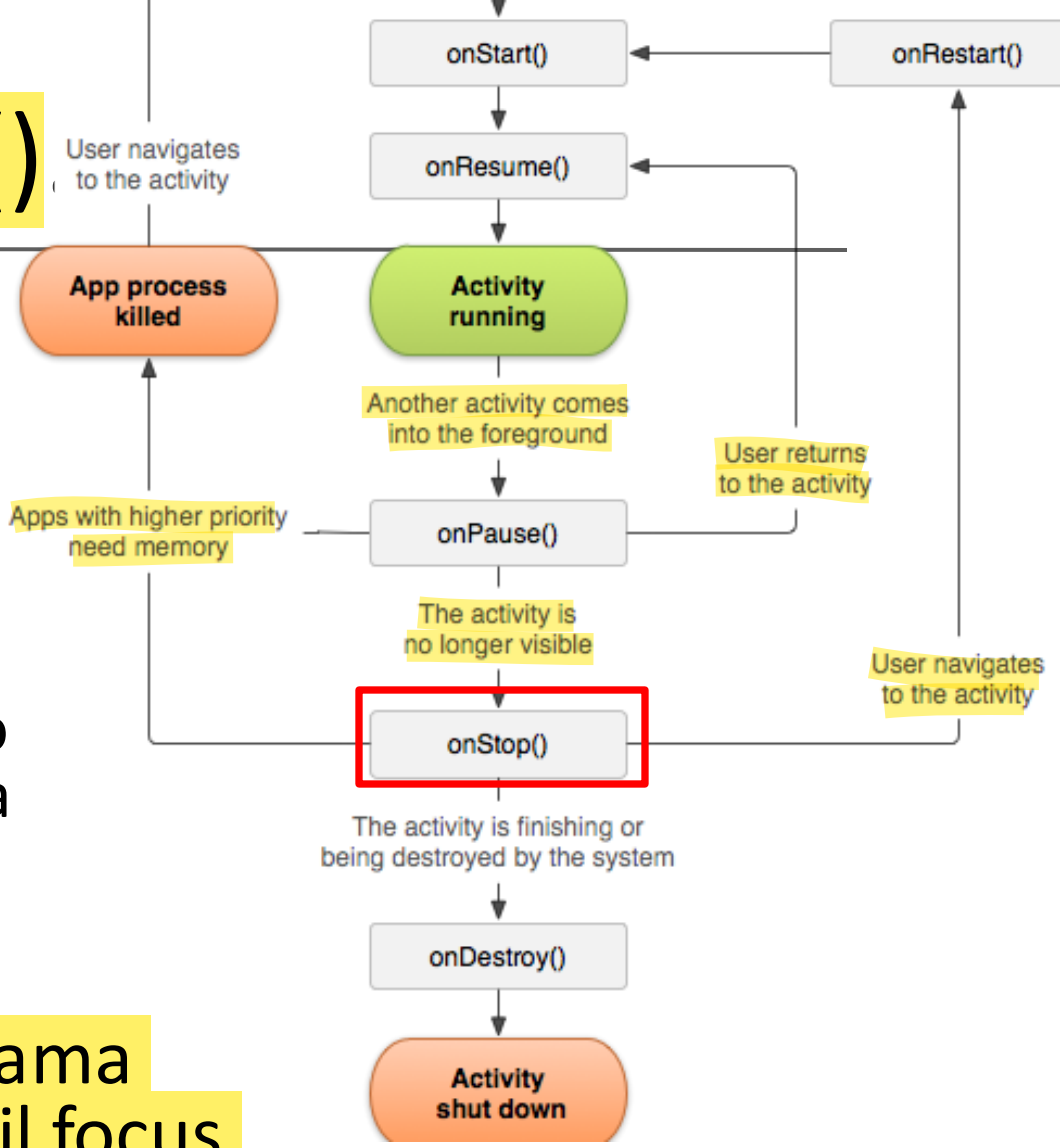
IMPORTANTE

- Il completamento del metodo onPause() non significa che l'attività lasci lo stato Paused!
- Piuttosto, l'attività rimane in questo stato fino a quando l'attività riprende o diventa completamente invisibile all'utente
- Se l'attività riprende, il sistema richiama ancora una volta il callback onResume()
  - Se l'attività ritorna dallo stato Paused allo stato Resumed, il sistema mantiene l'istanza Activity residente in memoria, richiamando quell'istanza quando il sistema richiama onResume()
  - In questo scenario, non è necessario re-inizializzare i componenti creati durante uno dei metodi di callback che hanno portato allo stato Resumed
- Se l'attività diventa completamente invisibile, il sistema chiama onStop()

IMPORTANTE

- Se l'attività ritorna allo stato Resumed dallo stato Paused, il sistema chiama ancora una volta il metodo onResume()
- Per questo motivo, dovresti implementare onResume() per inizializzare i componenti rilasciati durante onPause() ed eseguire qualsiasi altra inizializzazione che deve verificarsi ogni volta che l'attività entra nello stato Resumed

onStop()



- Il sistema chiama onStop() quando l'activity non è più visibile all'utente
  - Ciò può accadere perché l'attività viene distrutta, una nuova attività sta iniziando o un'attività esistente sta entrando in uno stato Resumed e copre l'attività interrotta
- In tutti questi casi, l'attività interrotta non è più visibile
- Il callback successivo che il sistema chiama è onRestart(), se l'attività sta riavendo il focus per interagire con l'utente, o onDestroy() se questa attività sta terminando completamente

Nota bene!

- Indipendentemente dall'evento di creazione in cui scegli di eseguire un'operazione di inizializzazione, assicurati di utilizzare l'evento del ciclo di vita corrispondente per rilasciare la risorsa
- Ovvero
  - Se inizializzi qualcosa dopo l'evento ON_START, rilascialo o terminalo dopo l'evento ON_STOP
  - Se inizializzi dopo l'evento ON_RESUME, rilascia dopo l'evento ON_PAUSE

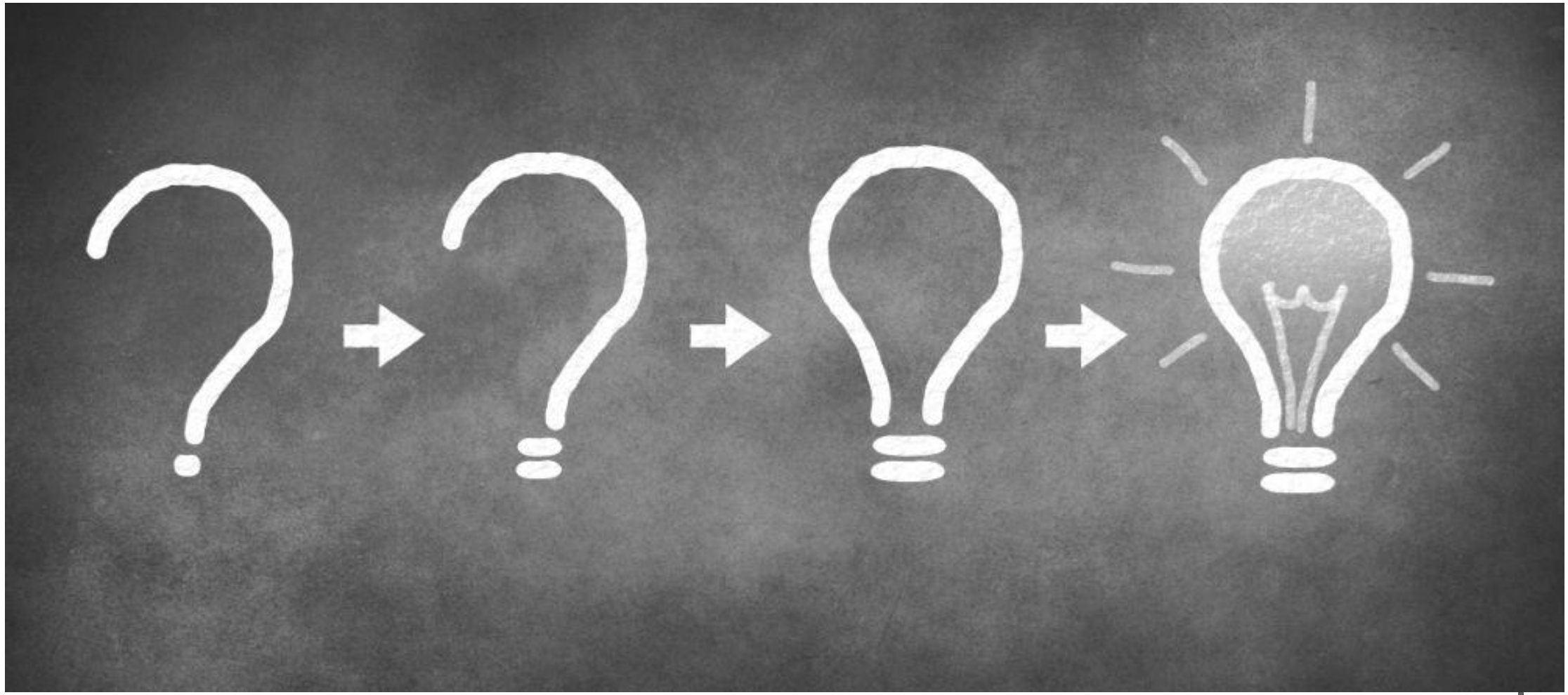
onDestroy()

- Il sistema richiama questo callback prima che un'activity venga distrutta
- Questo callback è quello finale che l'activity riceve
- onDestroy() viene in genere implementato per garantire che tutte le risorse di un'attività vengano rilasciate quando l'activity o il processo che la contiene viene distrutto

Importante

- onDestroy() viene chiamato prima che l'attività venga distrutta
- Il sistema richiama questa richiamata perché:
  - l'attività sta finendo (a causa dell'abbandono completo dell'attività da parte dell'utente, per esempio),
  - oppure il sistema sta temporaneamente distruggendo l'attività a causa di una **modifica della configurazione** (come la rotazione del dispositivo o la modalità multi-finestra)
- Il callback onDestroy() dovrebbe rilasciare tutte le risorse che non sono ancora state rilasciate da callback precedenti come onStop()

Domande?



Riferimenti

- https://developer.android.com/guide/components/activities/
- https://developer.android.com/guide/components/activities/activity-lifecycle
- https://developer.android.com/reference/kotlin/android/app/Activity