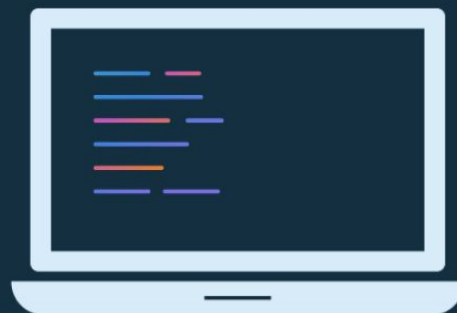




# Lezione 2.5: Ciclo di vita di Activity e Fragment



[Do the evolution - Pearl jam](#)



# Questa lezione

## Lezione 2.5: Ciclo di vita di Activity e Fragment

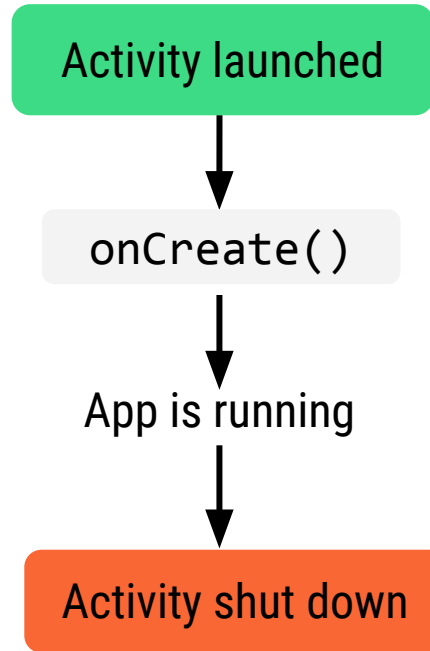
- Ciclo di vita delle Activity
- Logging
- Ciclo di vita dei Fragment
- Componenti lifecycle-aware

# Ciclo di vita delle Activity

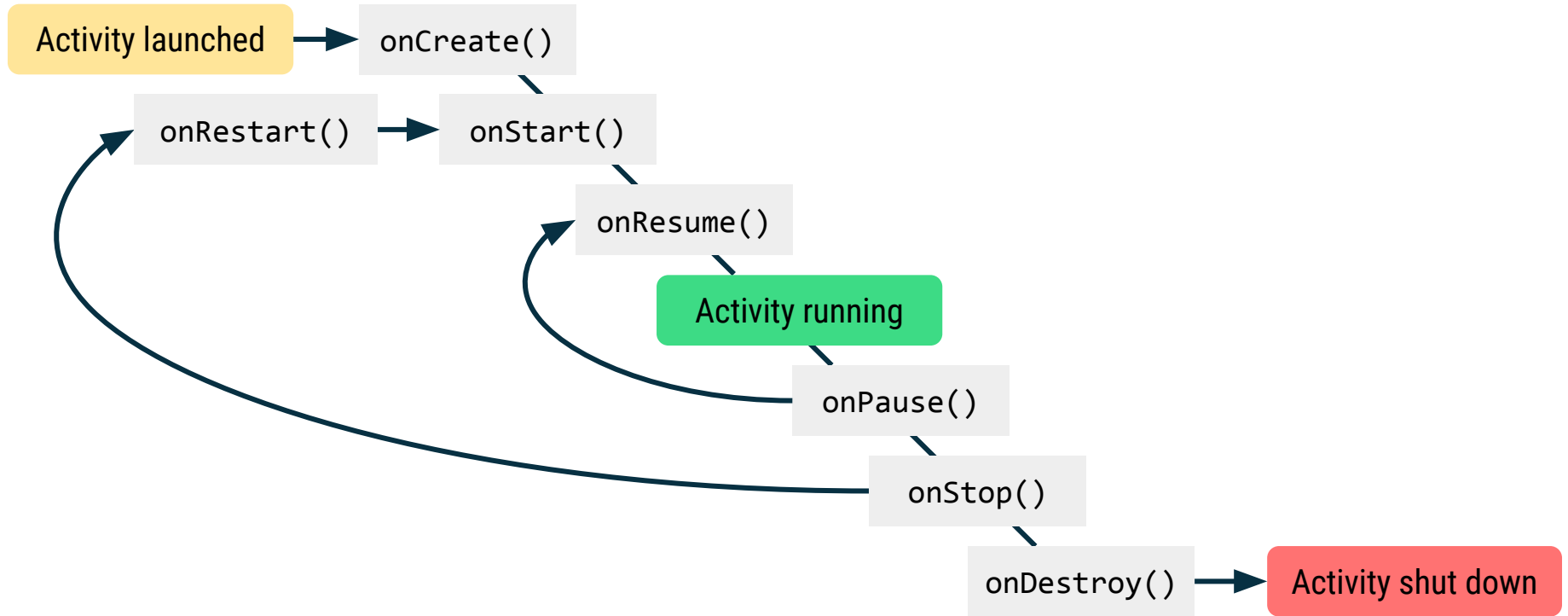
# Perché è rilevante

- Permette di conservare dati e stato se:
  - L'utente esce temporaneamente dall'app e poi ritorna
  - L'utente viene interrotto (ad esempio da una chiamata)
  - L'utente ruota il dispositivo
- Evita memory leaks e malfunzionamenti dell'app

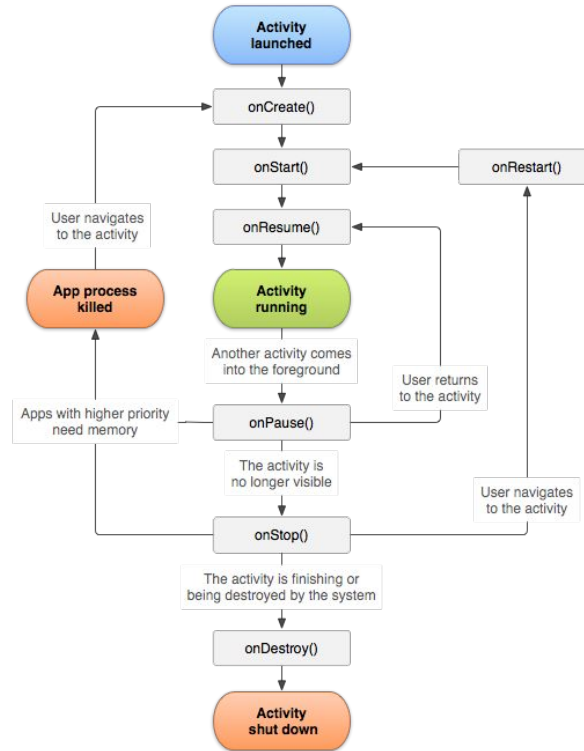
# Lifecycle semplificato



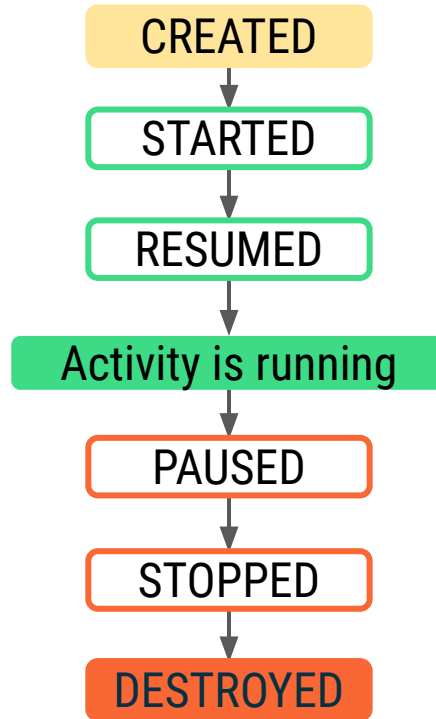
# Activity lifecycle



# Activity lifecycle



# Stati di un'Activity





# onCreate()

- L' Activity viene creata ed inizializzata
- Questa funzione di callback deve essere necessariamente implementata
- Si effettua l'inflating dell'UI ed altre eventuali operazioni logiche

# onStart()

- L'Activity diventa visibile all'utente
- Chiamata dopo:
  - `onCreate()`
  - **O**
  - `onRestart()` se l'activity era stata fermata (stopped)

# onResume()

- L'Activity acquisisce il focus:
  - L'utente può interagirci
- L'Activity rimane in questo stato finché il sistema non la mette in pausa

# onPause()

- L'Activity perde il focus (non è più in primo piano)
- L'Activity rimane visibile, ma l'utente non interagisce più in modo diretto
- E' il contrario di `onResume()`

# onStop()

- L'Activity non è più visibile all'utente (ma esiste ancora)
- Le risorse che non sono più necessarie vengono rilasciate
- Salva automaticamente ogni variabile dell'Activity

# onDestroy()

- L'Activity è in procinto di essere distrutta, a seguito di varie cause:
  - L'Activity ha terminato o è stata chiusa dall'utente
  - E' cambiata la configurazione (viene ruotato il terminale o il sistema chiude l'app per mancanza di memoria)
- Esegue una eliminazione finale delle risorse occupate
- È preferibile evitare di salvare eventuali dati in questo momento

# Riassunto degli stati di un'Activity

State	Callbacks	Description
Created	<code>onCreate()</code>	L'Activity viene inizializzata.
Started	<code>onStart()</code>	L'Activity è visibile all'utente.
Resumed	<code>onResume()</code>	L'Activity ha il focus.
Paused	<code>onPause()</code>	L'Activity non ha più il focus.
Stopped	<code>onStop()</code>	L'Activity non è più visibile.
Destroyed	<code>onDestroy()</code>	L'Activity viene distrutta.

# Salvare lo stato

L'utente si aspetta che l'UI rimanga uguale (mantenga i valori) dopo un cambiamento di configurazione o se l'app rimane in background.

- In questi casi l'Activity viene distrutta e ricreata, oppure è terminata e l'activity è fatta ripartire da zero
- E' possibile memorizzare piccole quantità di dati per ricostruire le activity:
  - Usiamo il `Bundle` fornito da `onSaveInstanceState()`
  - `onCreate()` riceve il `Bundle` come argomento quando l'Activity viene creata di nuovo



# Save state: esempi

Per salvare lo stato:

```
override fun onSaveInstanceState(outState: Bundle) {  
    outState.putString("key", "value")  
    super.onSaveInstanceState(outState)  
}
```

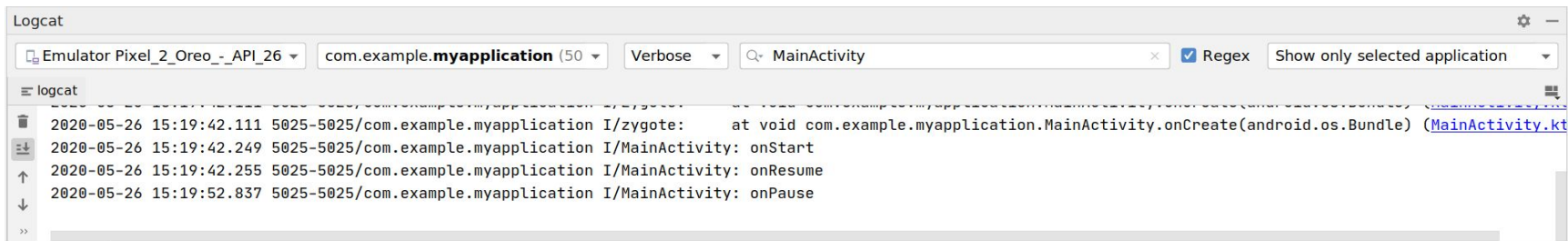
Per recuperarlo (dentro `onCreate` o dentro `onRestoreInstanceState`)

```
val myString = savedInstanceState?.getString("key")
```

# Logging

# Logging in Android

- Monitora il flusso di eventi o lo stato dell'app
- Si possono usare la classe built-in `Log` class o librerie esterne (es: [Timber](#))
- Esempio di chiamata ad un metodo: `Log.d(TAG, "Message")`

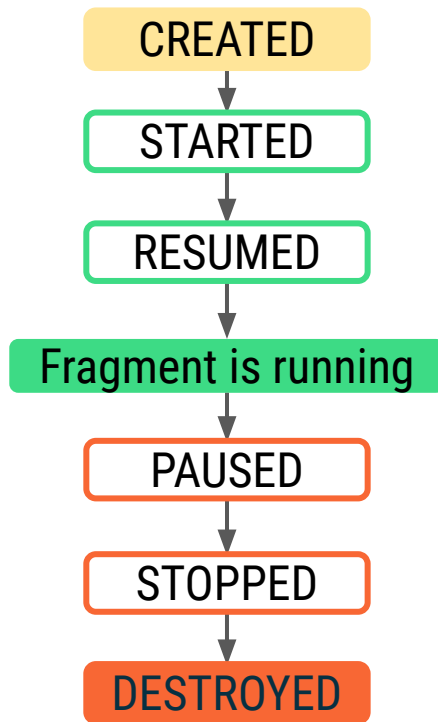


# Tipi di log

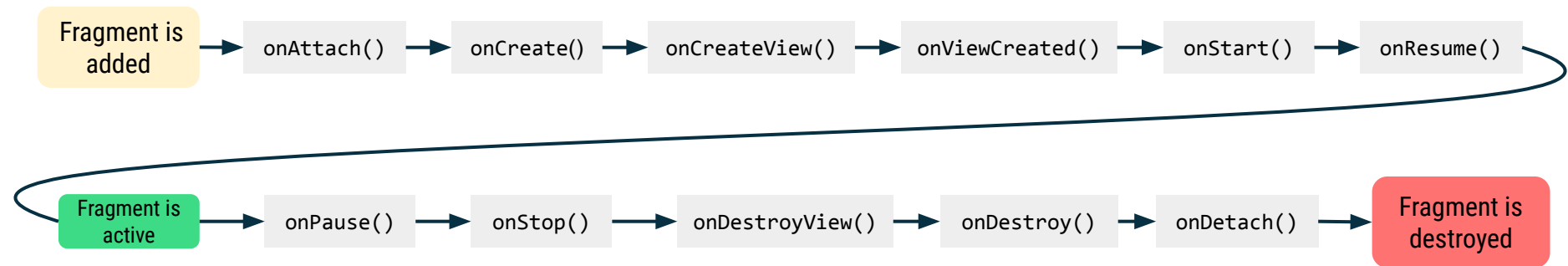
Priority level	Log method
Verbose	<code>Log.v(String, String)</code>
Debug	<code>Log.d(String, String)</code>
Info	<code>Log.i(String, String)</code>
Warning	<code>Log.w(String, String)</code>
Error	<code>Log.e(String, String)</code>

# Ciclo di vita dei Fragment

# Stati del Fragment



# Diagramma del ciclo di vita



# onAttach()

- Chiamato quando un Fragment viene agganciato ad un contesto
- Immediatamente prima di `onCreate()`



# onCreateView()

- Chiamato per creare la gerarchie delle View associata al Fragment
- Effettua l'inflating del layout e ritorna la View root (radice)

# onViewCreated()

- Viene chiamata quando la gerarchia delle View è stata completamente creata
- Qui si esegue ogni inizializzazione rimanente (ad esempio, ripristinare lo stato da un `Bundle`)

# onDestroyView() e onDetach()

- `onDestroyView()` viene chiamata quando la gerarchia di View del Fragment viene rimossa
- `onDetach()` è chiamato quando il Fragment non è più agganciato all'Activity ospite

# Riassunto degli stati dei Fragment

State	Callbacks	Description
Initialized	<code>onAttach()</code>	Il Fragment viene agganciato all'ospite.
Created	<code>onCreate()</code> , <code>onCreateView()</code> , <code>onViewCreated()</code>	Il Fragment è creato ed il layout viene inizializzato.
Started	<code>onStart()</code>	Il Fragment è creato ed è visibile.
Resumed	<code>onResume()</code>	Il Fragment ha il focus.
Paused	<code>onPause()</code>	Il Fragment non ha più il focus.
Stopped	<code>onStop()</code>	Il Fragment non è visibile.
Destroyed	<code>onDestroyView()</code> , <code>onDestroy()</code> , <code>onDetach()</code>	Il Fragment è rimosso dall'ospite.

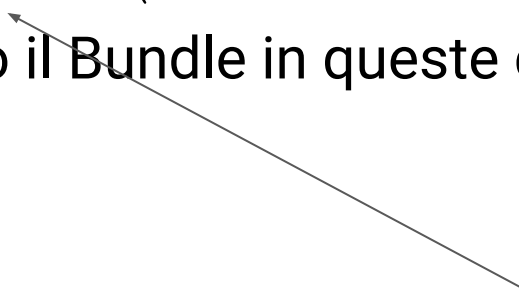
# Salvare lo stato dei Fragment

E' possibile preservare lo stato dell'UI in un Fragment all'interno di un Bundle:

- `onSaveInstanceState(outState: Bundle)`

Recupera i dati ricevendo il Bundle in queste chiamate di callback:

- `onCreate()`
- `onCreateView()`
- `onViewCreated()`



Ricordiamo di memorizzare pochi dati in questo modo. Se i dati sono molti useremo un DB.

# Componenti lifecycle-aware

# Componenti lifecycle-aware

A mano a mano che l'app diventa più complessa, inseriremo sempre più logica dentro i metodi relativi al ciclo di vita dell'Activity o di un Fragment.

Un approccio alternativo consiste nel creare componenti separate che contengono la logica e che sono *lifecycle-aware*, e cioè consapevoli del ciclo di vita della “componente” a cui sono agganciate.

# Componenti lifecycle-aware

Adattano il loro comportamento in base al ciclo di vita:

- Si usa la libreria `androidx.lifecycle`
- `Lifecycle` traccia lo stato di un'Activity o Fragment:
  - Memorizza lo stato corrente
  - Lancia gli eventi legati al ciclo di vita (quando cambia lo stato)



# LifecycleOwner

- Interfaccia che dichiara che una certa classe ha un Lifecycle
- Quando si usa occorre implementare il metodo `getLifecycle()`

Esempi: `Fragment` e `AppCompatActivity` sono di default implementazioni di `LifecycleOwner`

# LifecycleObserver

Se scriviamo una classe deputata a gestire gli eventi di lifecycle di un'Activity, allora dovrà implementare l'interfaccia `LifecycleObserver`:

```
class MyObserver : LifecycleObserver {  
    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)  
    fun connectListener() {  
        ...  
    }  
}
```

E dovrò registrare l'observer al particolare lifecycleOwner (ad esempio l'Activity)

```
myLifecycleOwner.getLifecycle().addObserver(MyObserver())
```