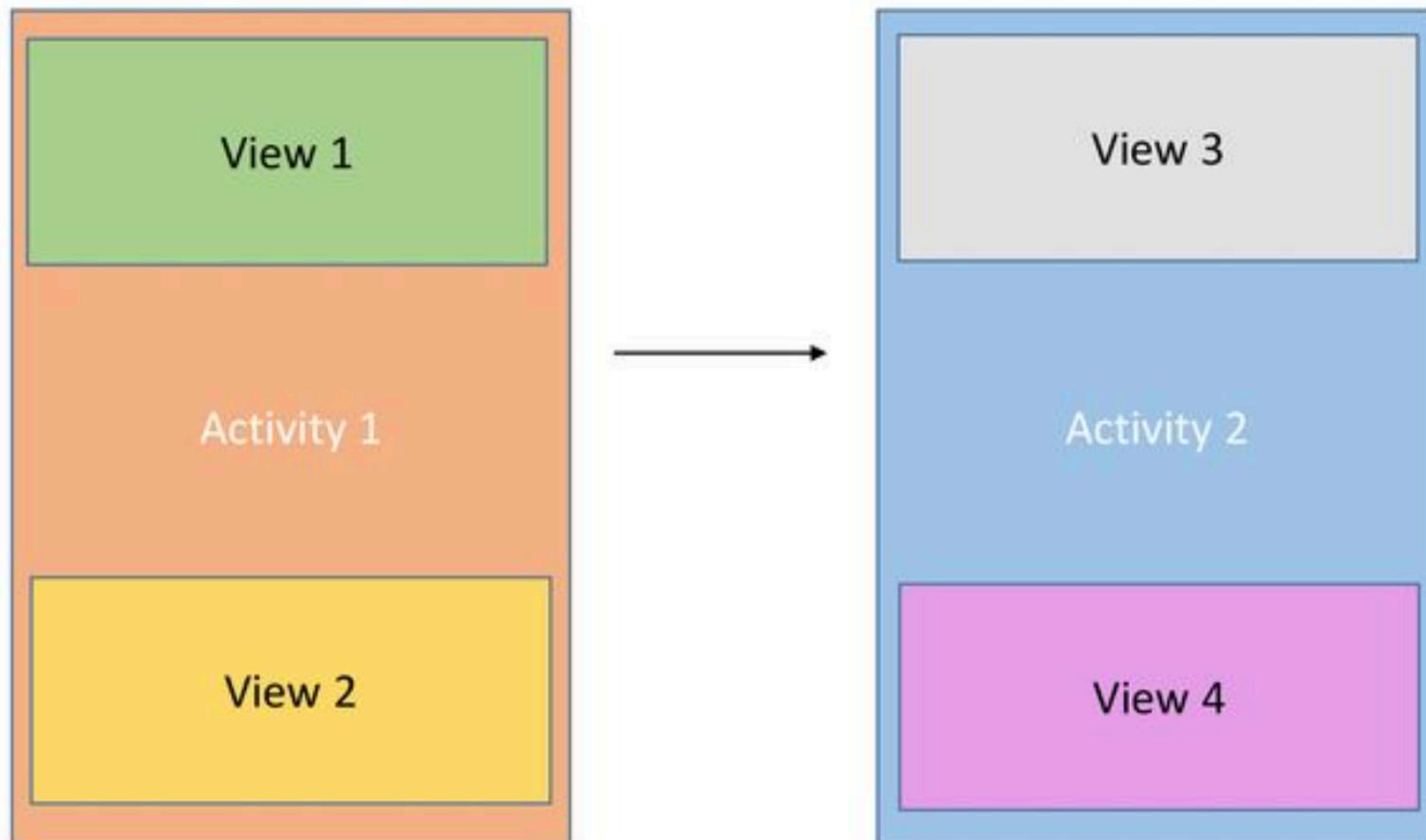
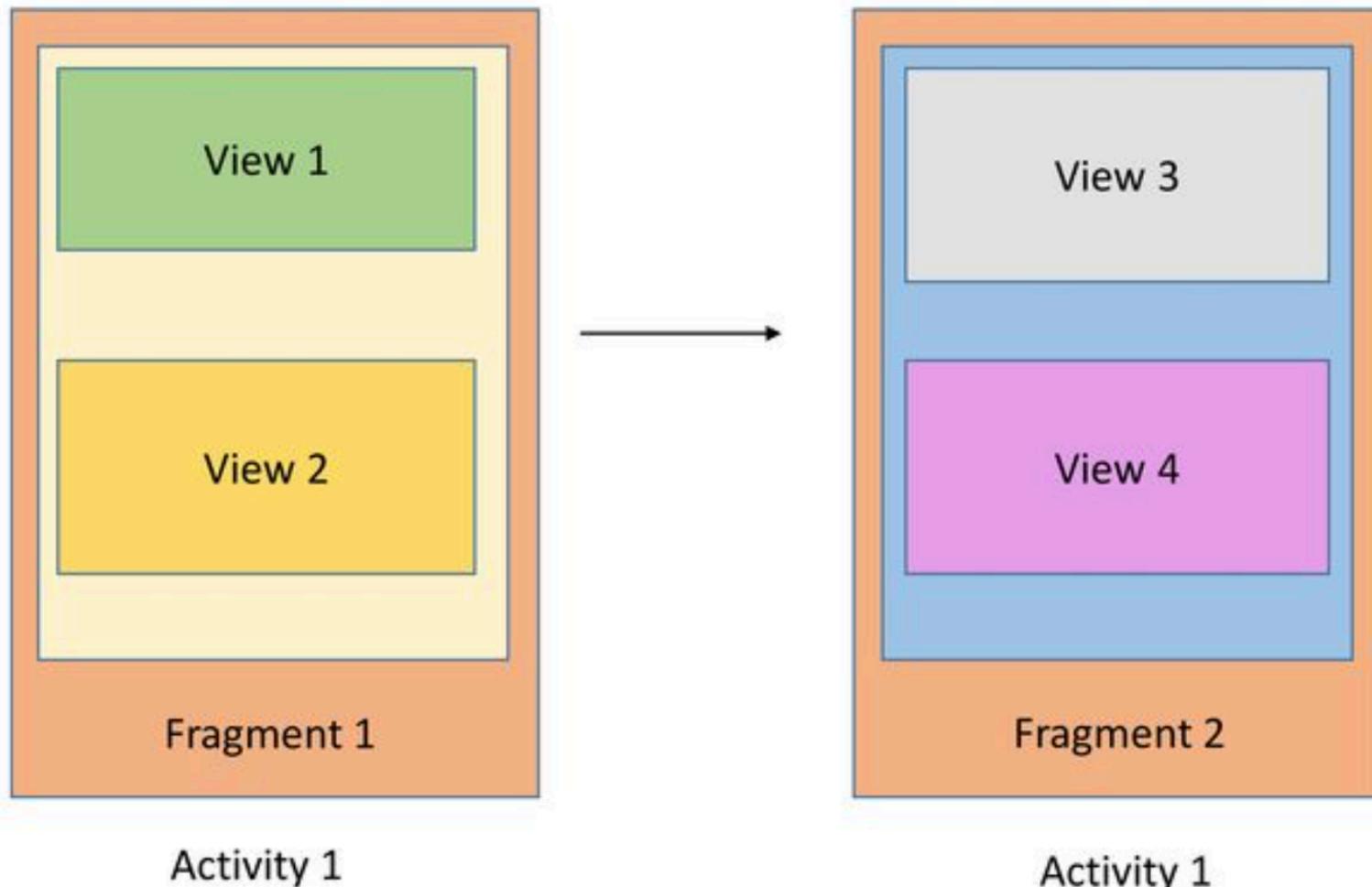


Fragment

Dalle Activity ...



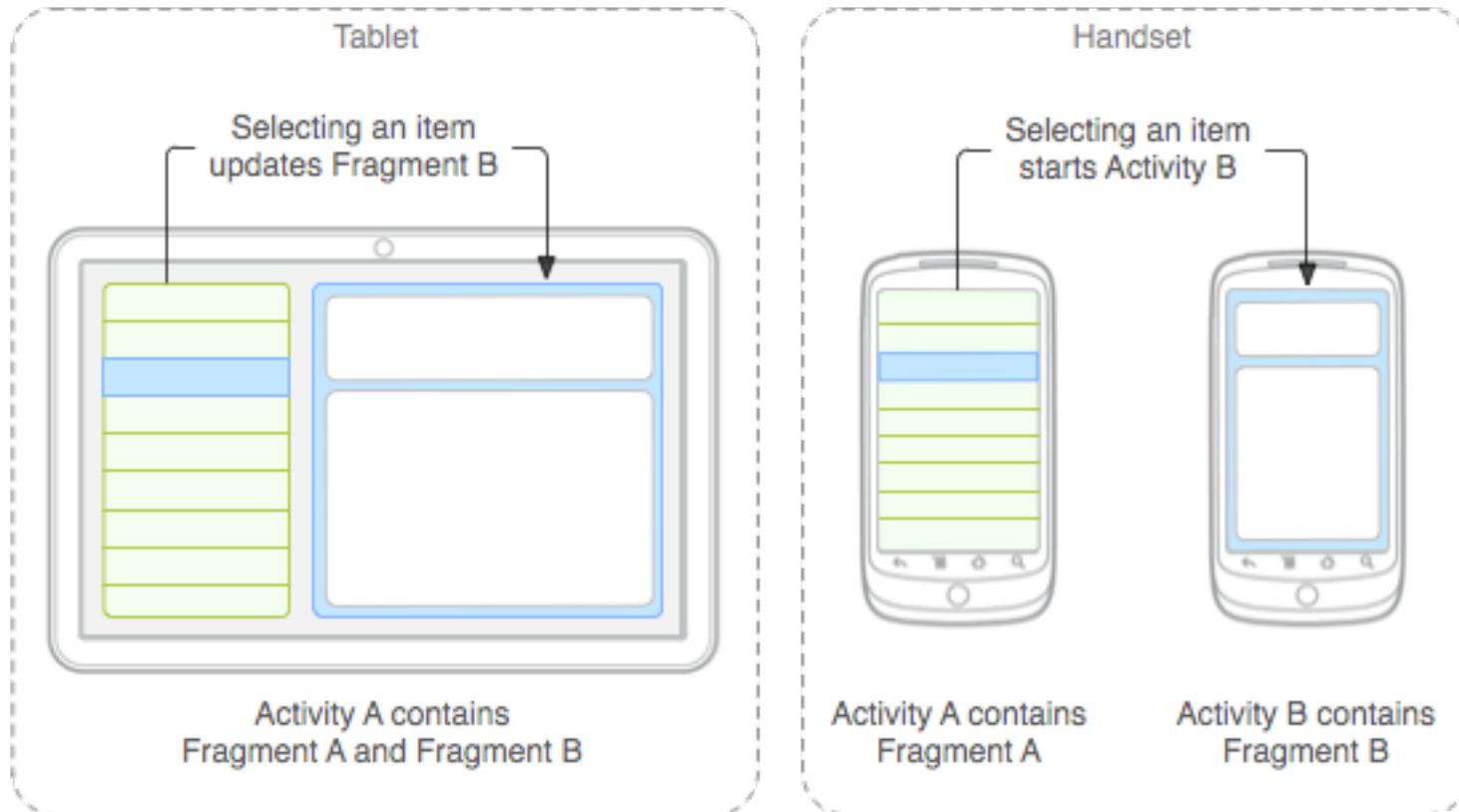
Ai Fragment



Fragment

- A Fragment represents **a behavior or a portion of user interface** in an Activity.
- **Caratteristiche**
 - sezione modulare che si può aggiungere e rimuovere
 - durante l'esecuzione di una activity
 - può gestire i propri eventi
 - vivono solo associati ad una activity
 - ha il proprio ciclo di vita
 - legato a quello dell'activity

Riuso di componenti



Fragment in xml

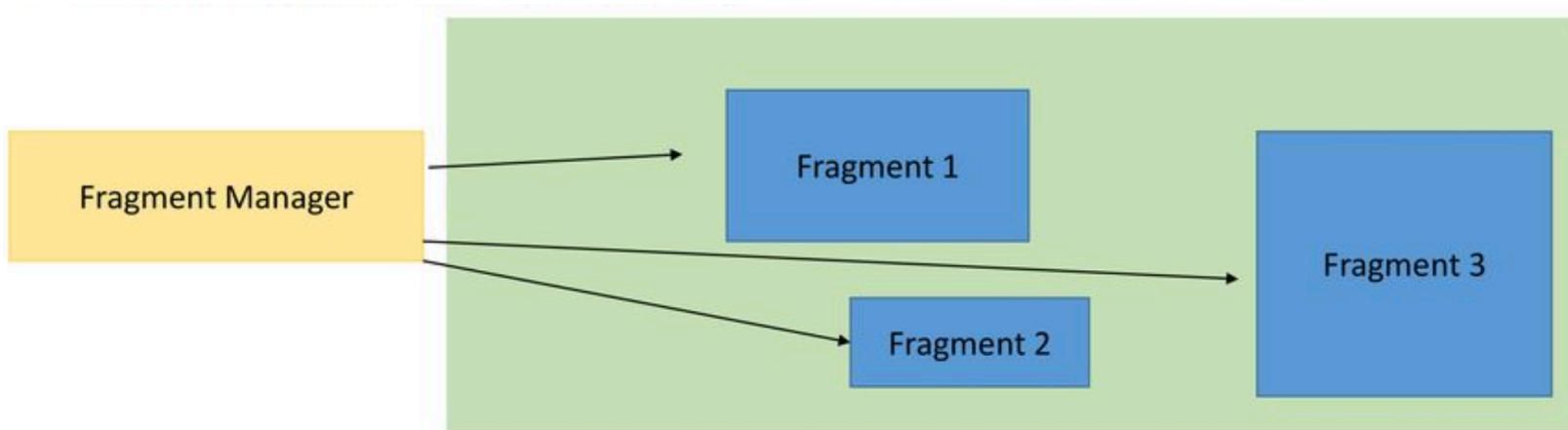
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
        android:id="@+id/list"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
        android:id="@+id/viewer"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
</LinearLayout>
```

Il Fragment viene **creato automaticamente** all'avvio dell'Activity.

Fragment Manager

Ogni **Activity** ha il proprio **Fragment Manager**, accessibile tramite:

- `getFragmentManager()` (per API < 28 o senza AndroidX)
- `getSupportFragmentManager()` (se si usa AppCompatActivity con AndroidX)
- Il Fragment Manager mantiene i riferimenti a **tutti i fragment** associati all'activity.



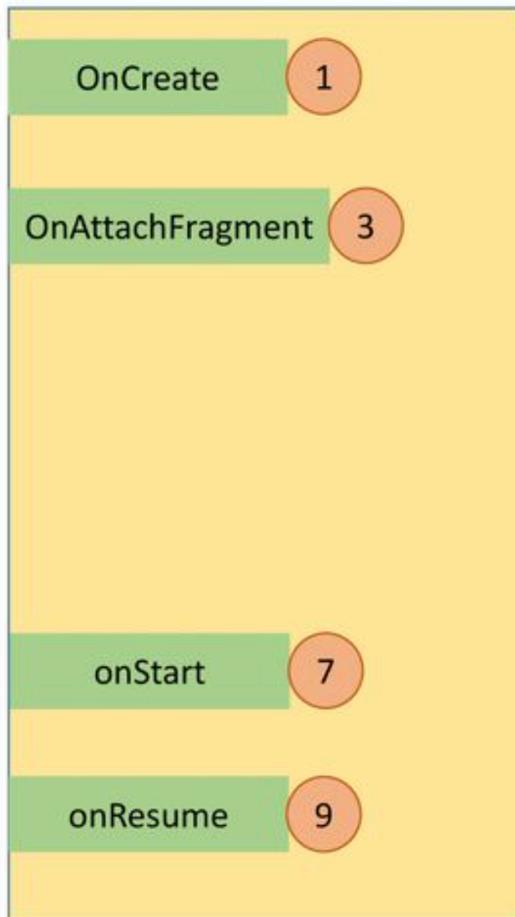
Fragment in java

```
public static class ExampleFragment extends Fragment {  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
                             Bundle savedInstanceState) {  
        // Inflate the layout for this fragment  
        return inflater.inflate(R.layout.example_fragment, container, false);  
    }  
}
```

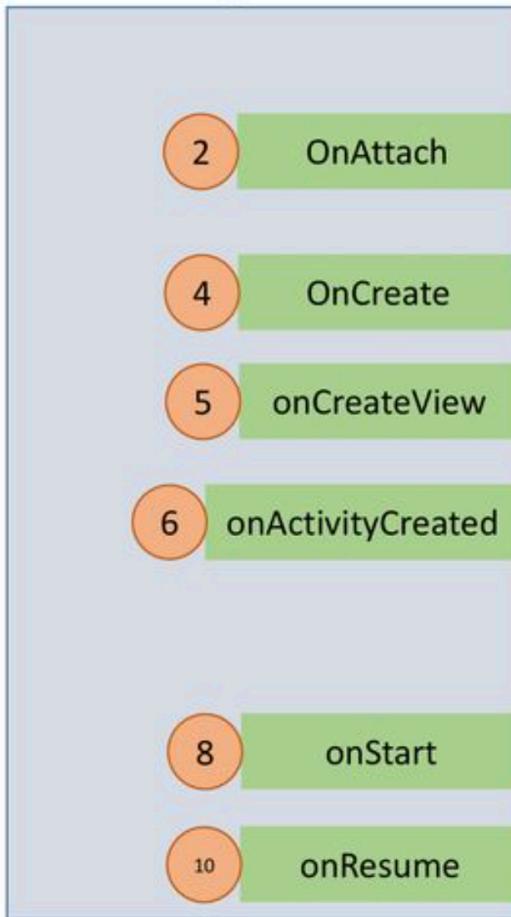
```
FragmentManager fragmentManager = getFragmentManager()  
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();  
  
ExampleFragment fragment = new ExampleFragment();  
fragmentTransaction.add(R.id.fragment_container, fragment);  
fragmentTransaction.commit();
```

Il ciclo di vita 1

Activity



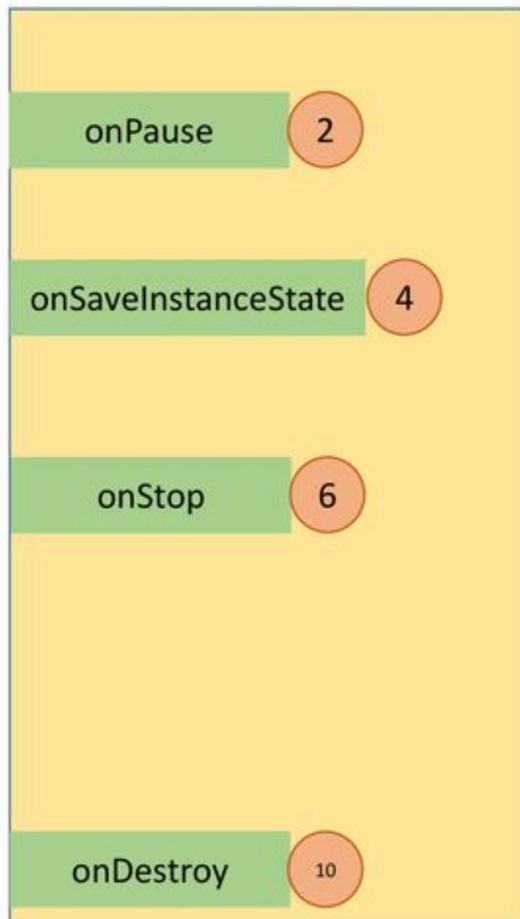
Fragment



`onAttach` is called after Fragment is associated with its Activity Gets a reference to the Activity object which can be used as Context.
`onCreateView`. You are expected to return a View Hierarchy for your fragment
`onActivityCreated` Called after Activity onCreate has completed execution Use this method to access/modify UI elements.

Il ciclo di vita 2

Activity



Fragment



onSaveInstanceState Use this to save information inside a Bundle object.

onDestoryView Called after the Fragment View Hierarchy is no longer accessible

onDestory Called after fragment is not used. It still exists as a java object attached to the Activity

onDetach Fragment is not tied to the Activity and does not have a View hierarchy

BackStack

- Il tasto back serve per navigare nella “storia” delle schermate
- La sequenza dei componenti da visualizzare e memorizzata nel Backstack
- Cambio di Activity
 - le schermate associate alle activity sono aggiunte automaticamente al backstack
- Cambio di Fragment
 - devo aggiungere manualmente la transizione al backstack
 - `transaction.addToBackStack(...)`

NAVIGATION

Dipendenze

```
dependencies {  
    def nav_version = "2.1.0"  
  
    // Java  
    implementation "androidx.navigation:navigation-fragment:$nav_version"  
    implementation "androidx.navigation:navigation-ui:$nav_version"  
  
    // Kotlin  
    implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"  
    implementation "androidx.navigation:navigation-ui-ktx:$nav_version"  
}
```

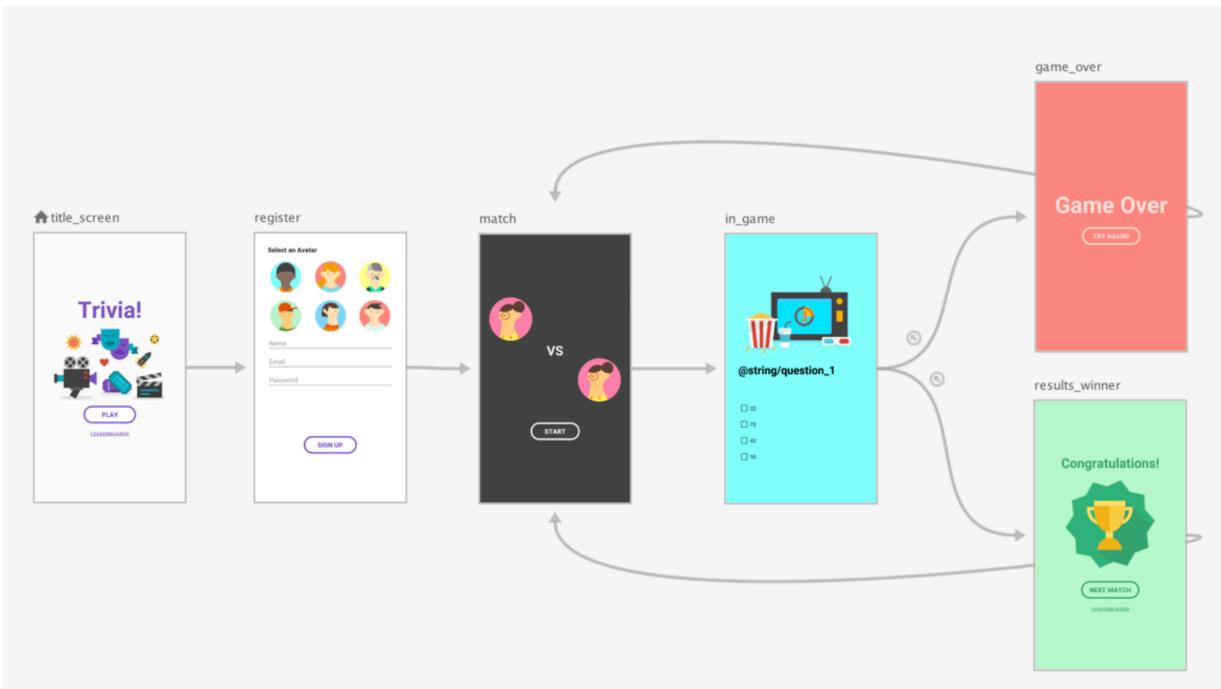
Navigation

Il componente di Navigazione è composto da tre parti fondamentali descritte di seguito:

- **Navigation graph**
 - Una risorsa XML che contiene tutte le informazioni relative alla navigazione
- **NavHost (NavHostFragment)**
 - Un contenitore vuoto che visualizza le destinazioni dal tuo navigation graph
- **NavController**
 - Un oggetto che gestisce la navigazione dell'app all'interno di un NavHost

Navigation Graph

- Un *navigation graph* è un file di risorse che contiene tutte le **destinazioni** e le **azioni**
 - Destinazioni: Fragment da visualizzare
 - Azioni: connessioni logiche tra le destinazioni
- Il grafo rappresenta tutti i percorsi di navigazione disponibili



NavController

- Oggetto che gestisce la navigazione dell'app all'interno di un NavHost.
 - Ogni NavHost ha il proprio NavController
- **Gradle Safe Args**
 - Plugin che genera oggetti semplici e classi builder che abilitano sia la navigazione che per il passaggio di dati tra le destinazioni.

Contesto	Metodo	Esempio	🔗
Da un Fragment	<code>findNavController()</code>	<code>findNavController().navigate(...)</code>	
Da una View	<code>view.findNavController()</code>	<code>button.findNavController().navigate(...)</code>	
Da un'Activity	<code>Navigation.findNavController(viewId)</code>	<code>Navigation.findNavController(R.id.nav_host_f ragment)</code>	

NavHost

- Un **Navigation Host** è un **contenitore UI** che ospita le destinazioni di navigazione
 - gestisce la navigazione tramite un NavController.

I navigation host devono:

- Gestire il salvataggio e il ripristino dello stato del proprio controller
- Chiamare Navigation.setViewNavController sulla loro vista radice
- Inoltrare gli eventi del pulsante "Indietro" del sistema al **NavController**,
 - NavController.popBackStack

VIEW MODEL E MVVM

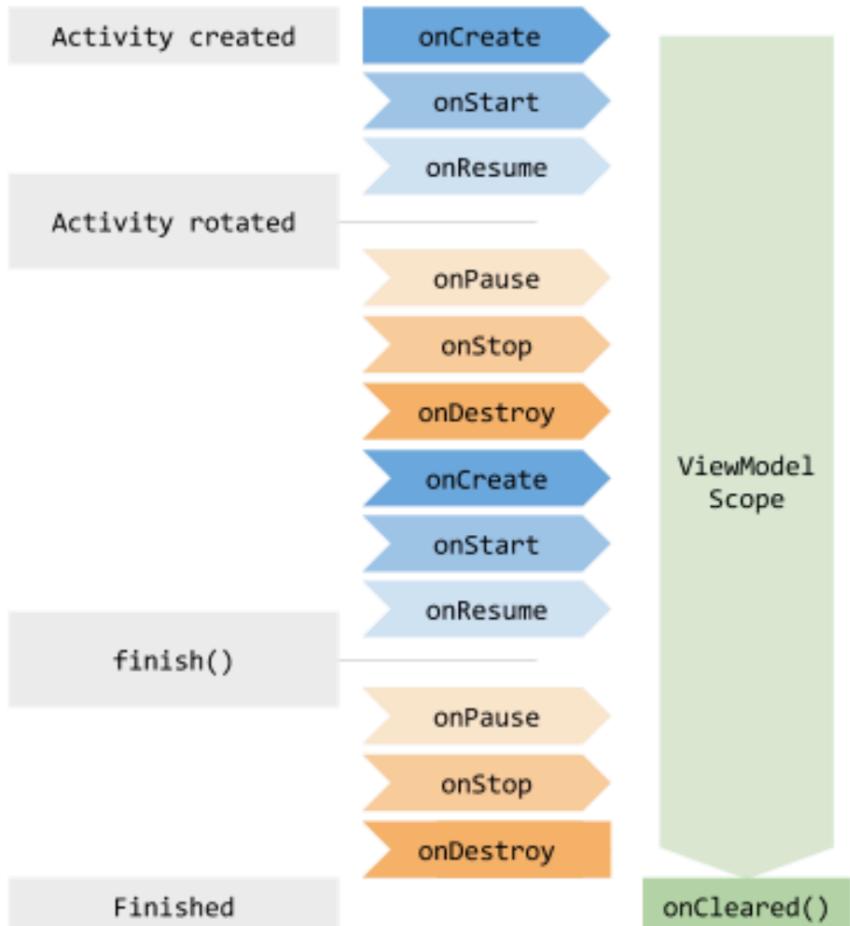
ViewModel

- Una classe che **mantiene lo stato e la logica di presentazione.**
- Vive più a lungo della Activity o Fragment e **sopravvive alle rotazioni.**
- Non ha riferimenti diretti alla View (UI).

```
public class CounterViewModel extends ViewModel {  
    private final MutableLiveData<Integer> counter = new MutableLiveData<>(0);  
  
    public LiveData<Integer> getCounter() {  
        return counter;  
    }  
  
    public void increment() {  
        counter.setValue(counter.getValue() + 1);  
    }  
}
```

ViewModel Scope

- Il ViewModel **non viene ricreato durante la rotazione**: è persistente.
- Viene distrutto **solo quando l'Activity finisce davvero** (`finish()`).
- `onCleared()` è l'equivalente di `onDestroy()` del ViewModel.



ViewModel pattern

- **Il ViewModel è un Scoped Singleton:**
 - un "singleton" nel contesto di uno specifico ViewModelStoreOwner come un Fragment o Activity.
 - Si chiama anche "Instance Cache per ciclo di vita" o "Lifecycle-scoped Singleton".
- **Non è un Singleton classico:**
 - Non c'è una sola istanza globale.
 - Ogni Activity o Fragment può avere la sua copia separata.

LiveData

- Oggetto **osservabile** che notifica automaticamente la UI quando cambia.
- È **readonly**: la View può osservarlo ma **non può modificarlo**.
- Viene usato nella View (Activity o Fragment) per reagire ai cambiamenti.

```
viewModel.getCounter().observe(getViewLifecycleOwner(), count -> {  
    textView.setText(String.valueOf(count));  
});
```

MutableLiveData

- Una sottoclasse di LiveData che può essere **modificata** (scrittura).
- Va usata **solo dentro il ViewModel**.
- Serve per cambiare i dati osservabili dalla View.

```
private final MutableLiveData<String> name = new MutableLiveData<>();  
public LiveData<String> getName() {  
    return name;  
}
```

```
public void increment() {  
    counter.setValue(counter.getValue() + 1);  
}
```

MVVM

■ 1. Model

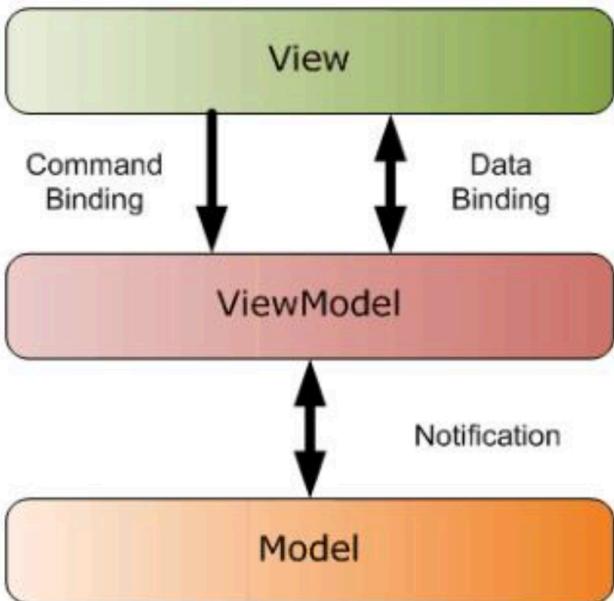
- Contiene i **dati puri** e la logica di accesso ai dati.
- Può includere repository, API, database, ecc.
- Non conosce né la View né il ViewModel.

■ 2. ViewModel

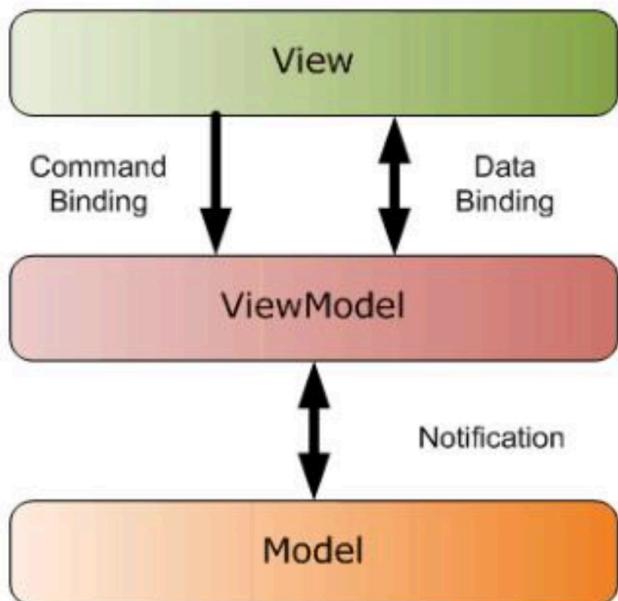
- Fa da **ponte tra Model e View**.
- Non ha riferimenti alla Activity o Fragment → è indipendente dalla UI.

■ 3. View

- È la UI: Activity, Fragment, XML layout.
- Osserva i dati del ViewModel (es. LiveData) e li mostra.
- Reagisce agli input dell'utente e notifica il ViewModel.



MVVM



Dalla	A	Tipo	Significato
View → ViewModel	Command Binding	Esempio: <code>viewModel.onButtonClick()</code>	
ViewModel → View	Data Binding	Esempio: <code>LiveData<String></code> aggiornato → aggiorna <code>TextView</code>	
ViewModel ↔ Model	Notification	Esempio: <code>viewModel.loadUser()</code> → Model restituisce i dati → <code>LiveData</code> aggiorna la View	

