FRAGMENTS

Panoramica

- I Fragment costituiscono senz'altro uno dei più importanti elementi per la creazione di una interfaccia utente Android moderna.
- Sono stati introdotti a partire da Android 3.0 e sono a grandi linee da intendere come delle mini-activity: visualizzano una certa interfaccia grafica e svolgono determinate funzioni.
- È bene sottolineare come essi non estendano le Activity. Generalmente sono quest'ultime ad estendere la classe Fragment (o alcune sottoclassi) per definirne meglio il comportamento.

Implementazione

- È possibile combinare più fragment in una singola Activity per creare un'interfaccia utente a più riquadri ed è anche possibile riutilizzarli altrove.

- Inizialmente erano disponibili facendo uso della libreria di supporto loro dedicata:

```
import android.support.v4.app.Fragment;
```

- Libreria che però è ormai deprecata ed è stata rimpiazzata dal package “androidx”:

```
import androidx.fragment.app.Fragment;
```

Utilizzo

- Offrono la possibilità di creare interfacce utente composte da porzioni riutilizzabili e alternabili senza la necessità di distruggere l'Activity.

- Un Fragment ha bisogno di un'Activity che gli faccia da contenitore ed ogni Activity può avere al suo interno più fragment (e quindi più mini-activity) e questi possono essere sostituiti da altri fragment mentre l'applicazione è attiva.

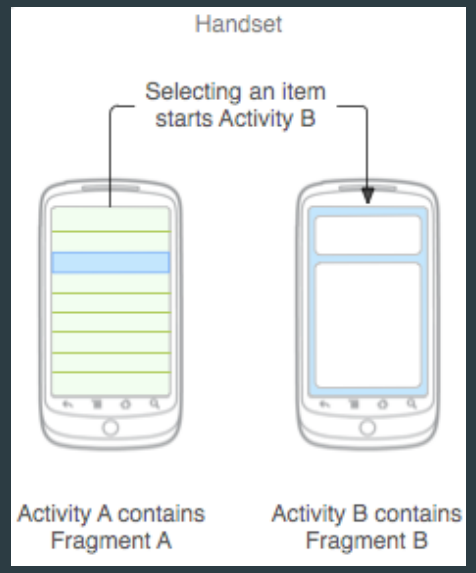
- Una struttura del genere facilita la gestione del ciclo di vita di un'Activity, visto che quest'ultima non cambia ma si alternano solo i Fragment.

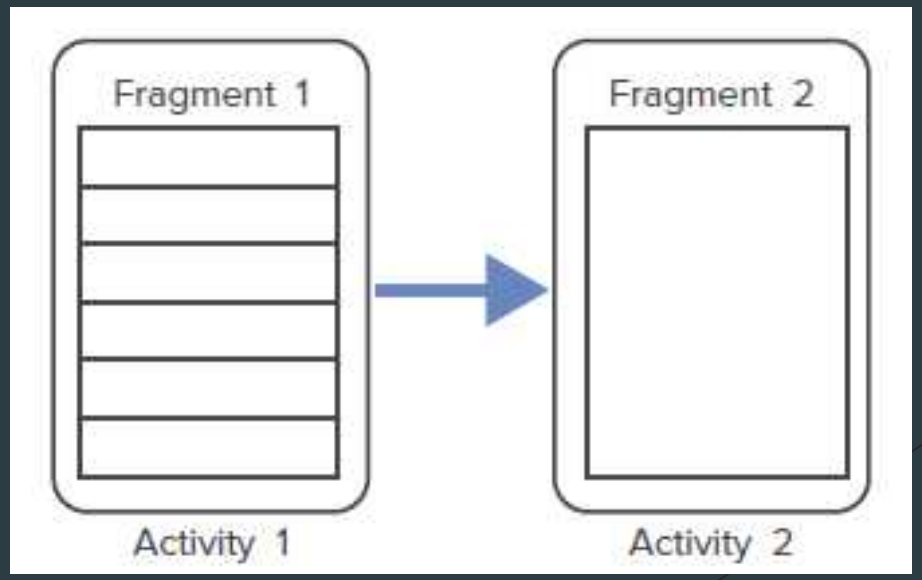
- D'altro canto però aumenta la complessità considerando che i Fragment hanno un proprio ciclo di vita piuttosto articolato.

Punto di Forza

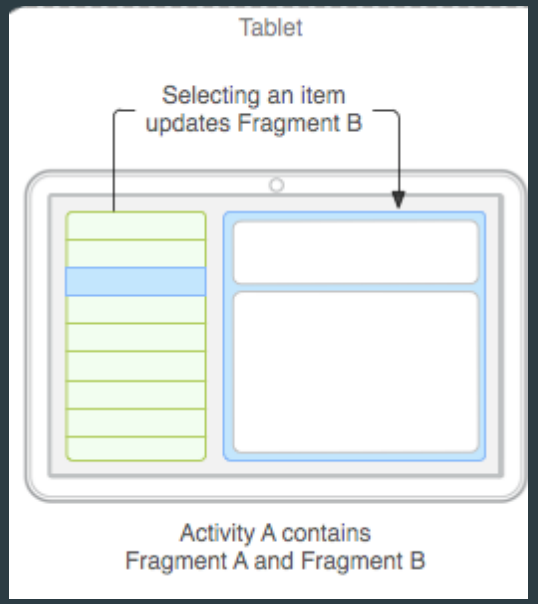
- Supponiamo di avere due Fragment, uno che contiene una ListView ed un altro alcune TextView, e di avere un'applicazione che li implementi.

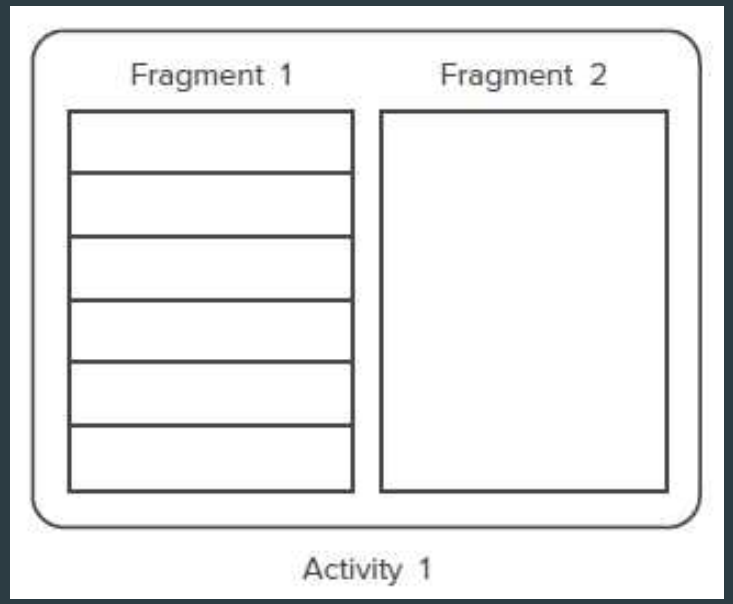
- Andando a visualizzare l'applicazione su un dispositivo Handset, o comunque sia ad orientamento verticale, il primo fragment verrà inserito in un'activity e il secondo in un'altra.





- Visualizzando l'applicazione in orizzontale o ad esempio su un Tablet si nota come entrambi i fragment possano essere visualizzati all'interno della stessa Activity.





- Si capisce quindi come i fragment costituiscano un modalità versatile per creare le interfacce utente di applicazioni Android.

Metodi di Definizione

- Vi sono due modi per definire un Fragment: dichiararandolo direttamente nel layout, nonchè metodo più semplice, oppure tramite codice.

- Per aggiungere un fragment al layout di un'activity si utilizza il tag <fragment>

```
<fragment
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/fragment"
    android:name="com.example.fragment.Blankfragment"/>
```

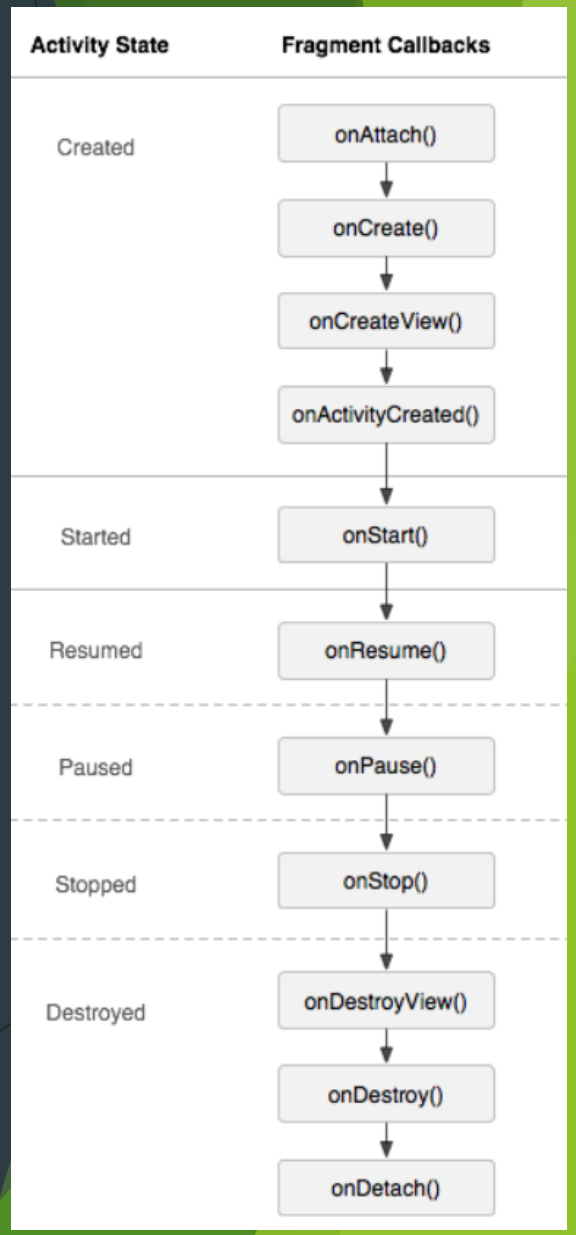
Lifecycle

Lifecycle: created

Per creare un fragment è necessario creare una sottoclasse di *Fragment.* La classe *Fragment* contiene metodi simili a quelli di un'activity, come ad esempio onCreate(), onStart(), onPause(), onStop().

*P*er creare un fragment è necessario implementare i seguenti metodi:

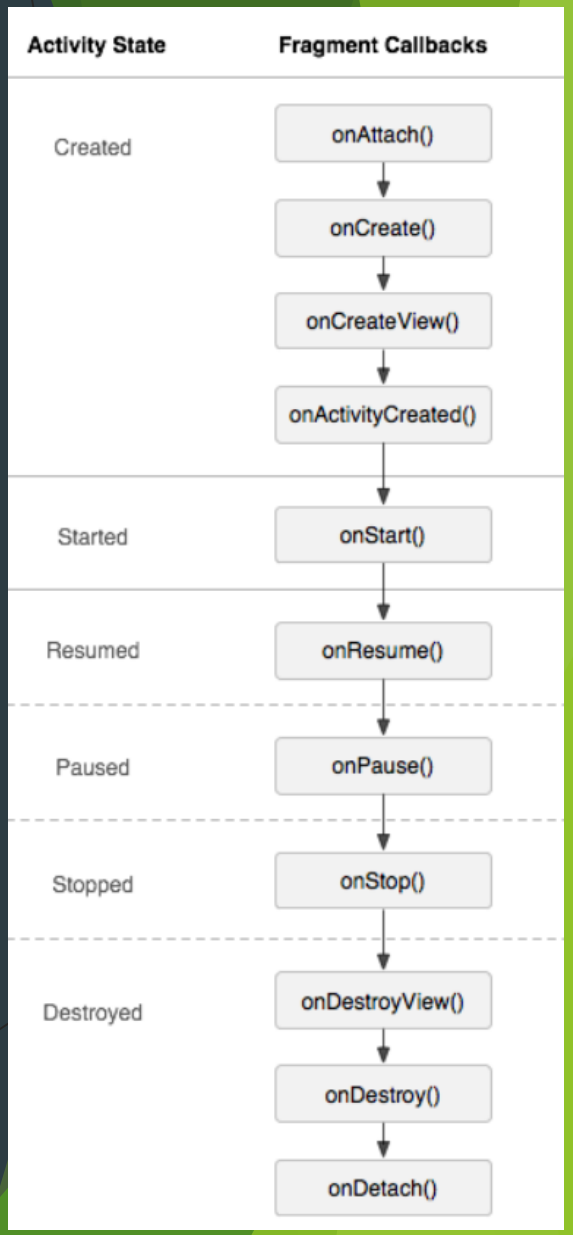
- ***onAttach():*** questo metodo viene chiamato quando il frammento è stato associato con la Activity.
- ***onCreate():*** il sistema chiama tale metodo durante la creazione iniziale del fragment. Al suo interno è necessario inizializzare i componenti essenziali del fragment.



Lifecycle: created

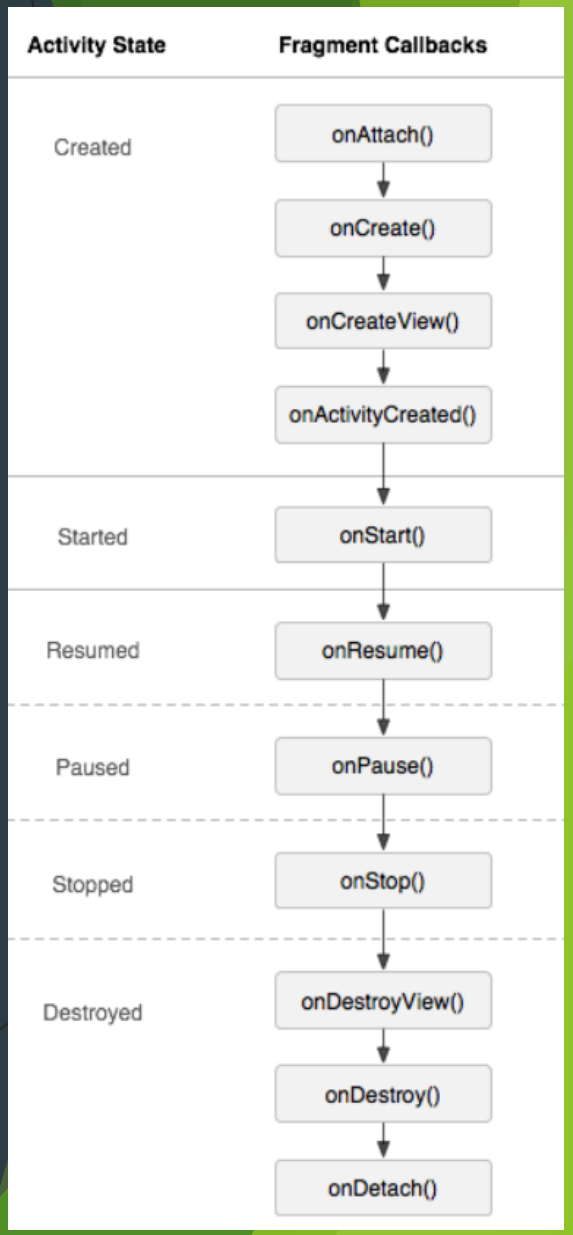
- ***onCreateView():*** questo metodo viene chiamato dal sistema quando il frammento deve disegnare la sua interfaccia utente per la prima volta. Questo metodo infatti restituisce un oggetto *View*, il quale rappresenta la radice del layout del frammento (struttura gerarchica associata al frammento).

```
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);
    }
}
```



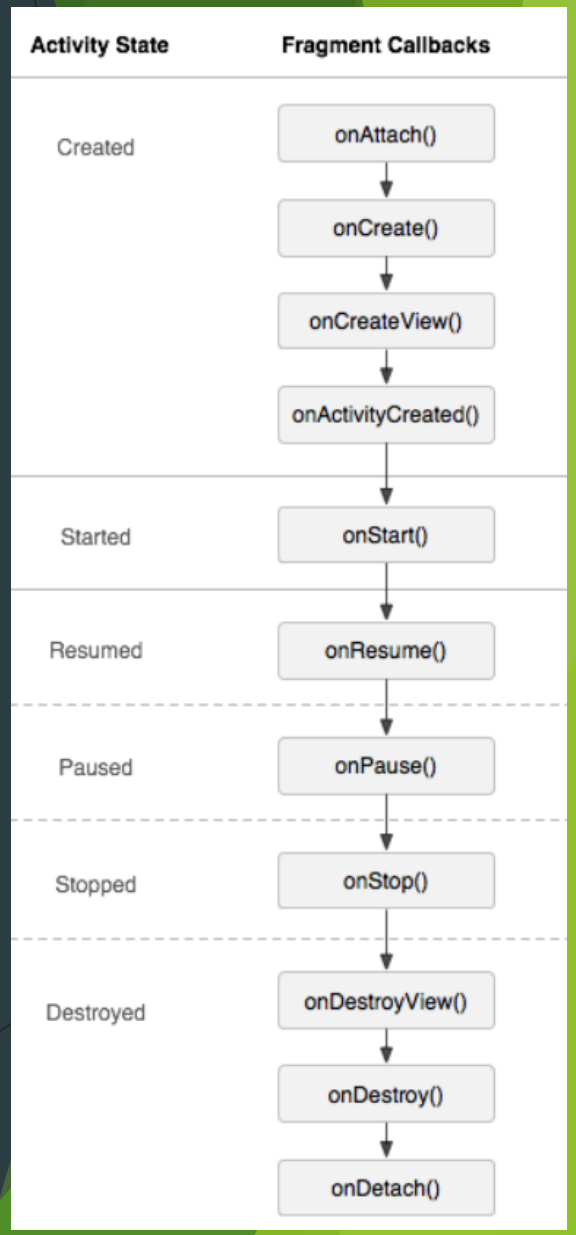
Lifecycle: created

- ***onActivityCreated():*** questo metodo segnala al fragment che il metodo onCreate() della sua activity è ritornato e la gerarchia della vista di questo frammento è stata istanziata.



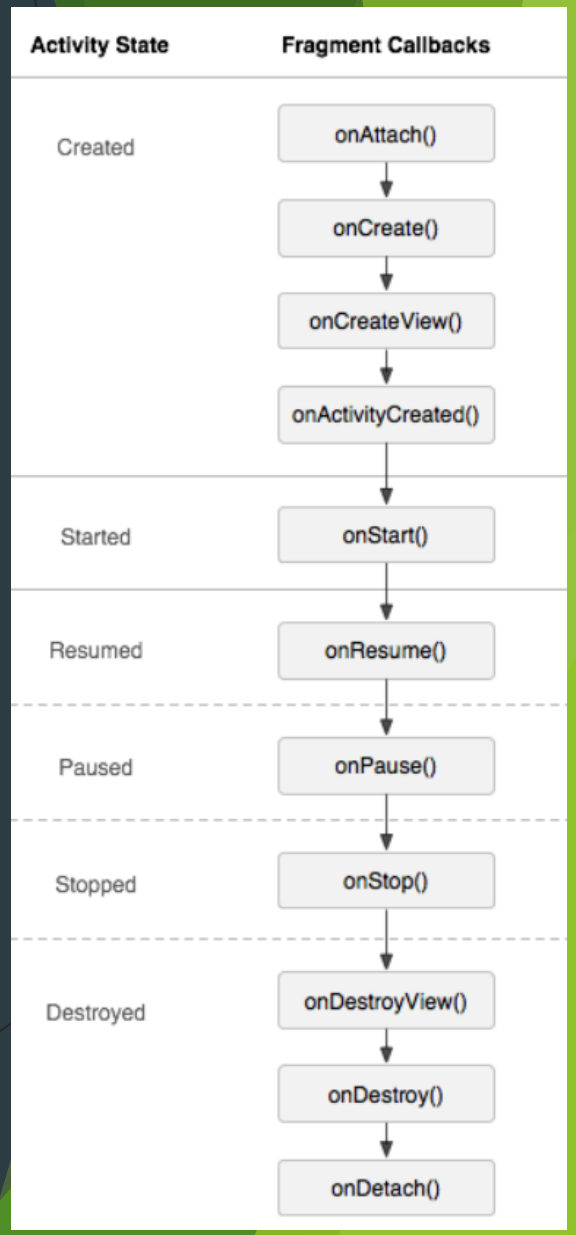
Lifecycle: started and resumed

- ***onStart():*** con questo metodo il fragment diventa **visibile** all'utente.È il momento in cui si possono attivare funzionalità e servizi che devono offrire informazioni all'utente.
- ***onResume():*** permette all'utente di interagire con il fragment.



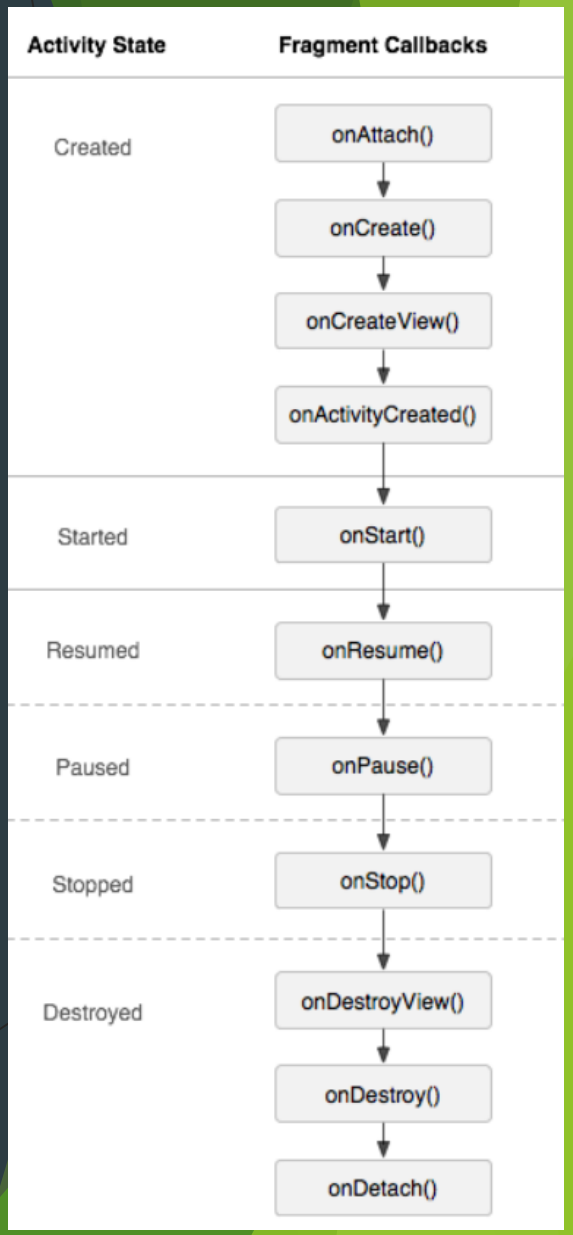
Lifecycle: paused

- ***onPause():*** questo metodo viene chiamato dal sistema come indicazione che l'utente sta lasciando il frammento (il che non vuol dire necessariamente che il frammento sta per essere distrutto). In questa fase, in generale, è necessario eseguire il **commit** di eventuali modifiche che devono essere mantenute oltre la sessione.



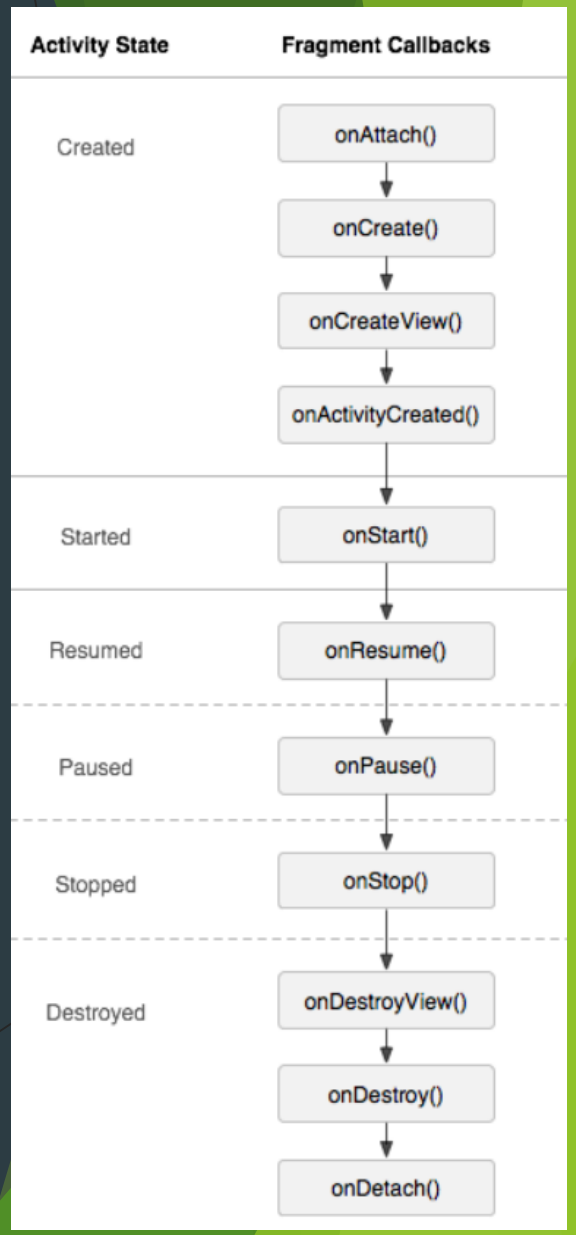
Lifecycle: stopped

- ***onStop():*** con questo metodo il fragment diventa non visibile. L'attività dell'host è stata interrotta o il frammento è stato rimosso dall'attività ma aggiunto al back stack.Un frammento arrestato è ancora **vivo** ma non è più visibile all'utente e viene ucciso se l'attività viene interrotta. Tutte le informazioni sullo stato e sui membri vengono conservate dal sistema).



Lifecycle: destroyed

- ***onDestroyView():*** questo metodo viene invocato quando la vista gerarchicamente associata con il fragment viene rimossa.
- ***onDestroy():*** questo metodo viene invocato per fare un clean-up finale dello stato del fragment.
- ***onDetach():*** questo metodo viene invocato quando il fragment viene dissociato dall'activity.



Activity lifecycle VS Fragment lifecycle

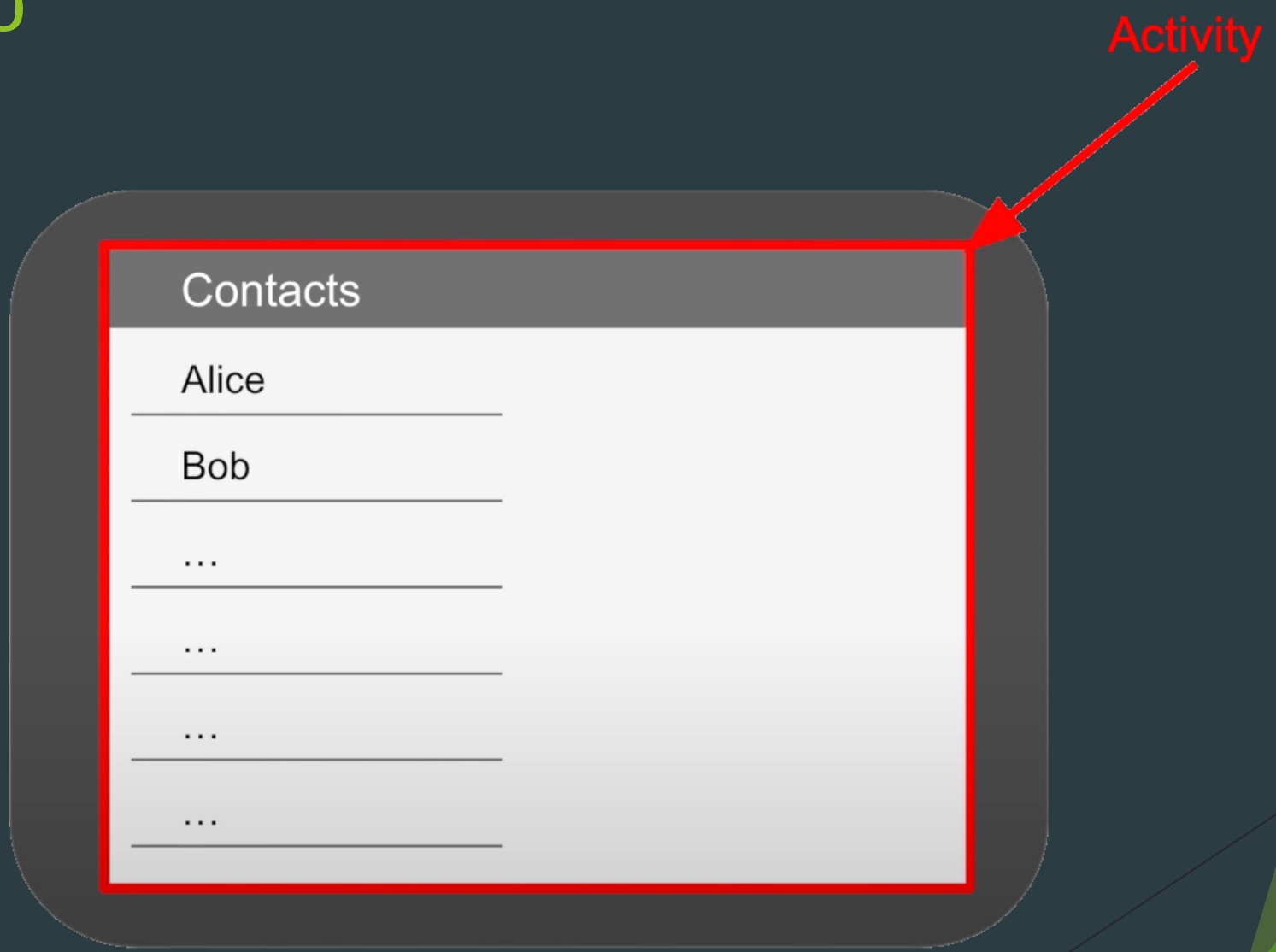
- La differenza più significativa tra il ciclo di vita di un’activity e quello di un fragment è costituita nella modalità di memorizzazione nel rispettivo back stack.
- Un'attività viene inserita nel back stack di attività gestite dal sistema quando viene interrotta, per impostazione predefinita (in modo che l'utente possa tornare indietro con il pulsante *Back).*
- Un frammento viene inserito in uno stack gestito dall'attività host, solo quando si richiede esplicitamente che l'istanza venga salvata chiamando ***addToBackStack****()* durante una transazione che rimuove il frammento.

Build a flexible UI

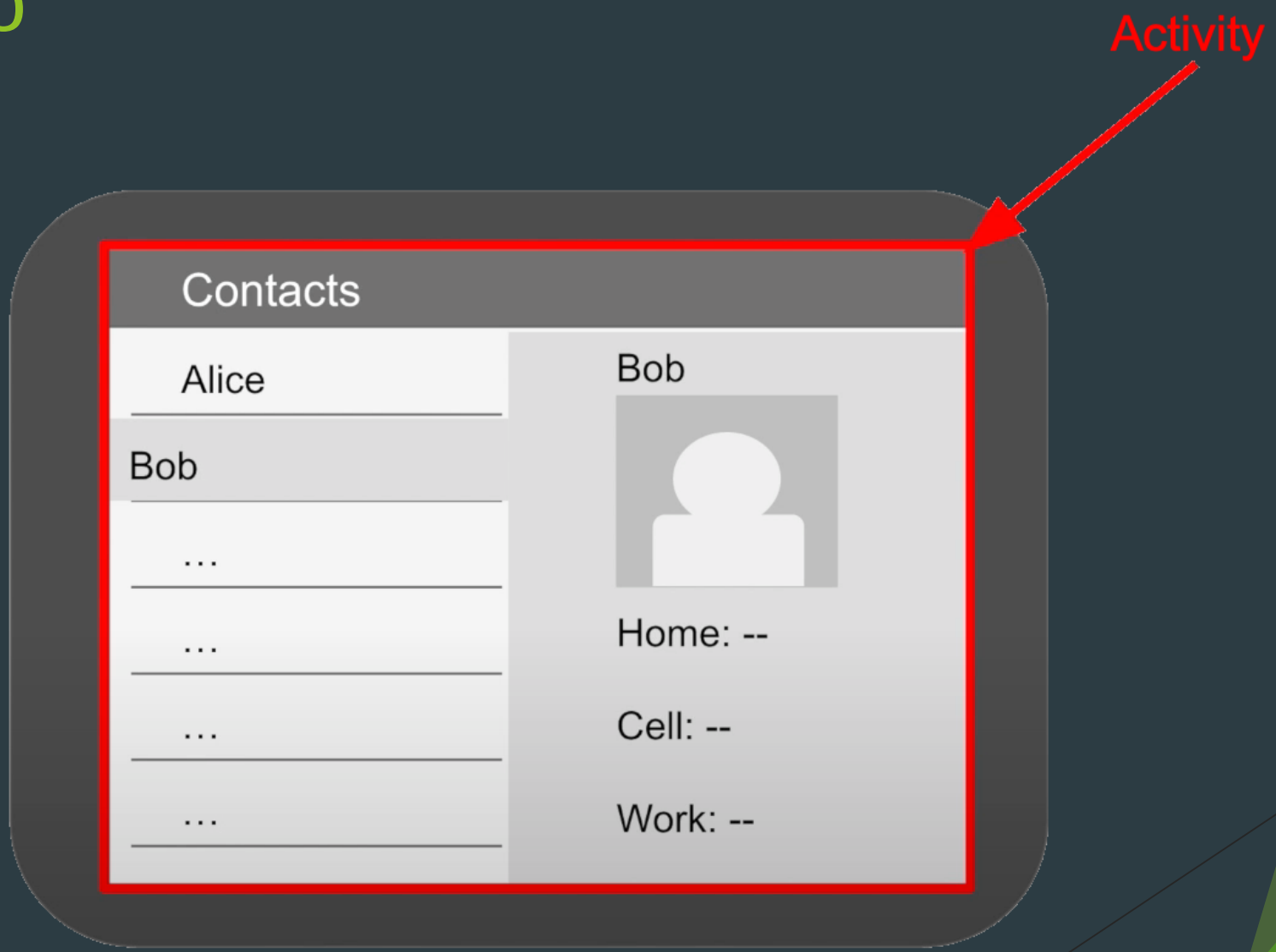
Perché usare i fragments?

- Durante l'utilizzo di un dispositivo Android si preferirebbe utilizzare al massimo lo spazio a disposizione, evitando cambi di layout o scrollare di la pagina.
- Ciò non significa che c'è bisogno di far entrare tutto in un'unica schermata rendendo l'applicazione difficilmente navigabile
- L'utilizzo dei Fragment fa si che questo problema possa essere agirato molto facilemnte, poiché si possono combinare in diverse configurazioni di layout, soprattuto in modalità landscape e su tablet, dove:
  - abbiamo più spazio per gestire le nostre schermate
  - l'utente, invece che vedere la stessa schermata con dello spazio inutilizzato, preferirebbe vedere la schermata ottimizzata per il suo scopo in tutto lo spazio disponibile

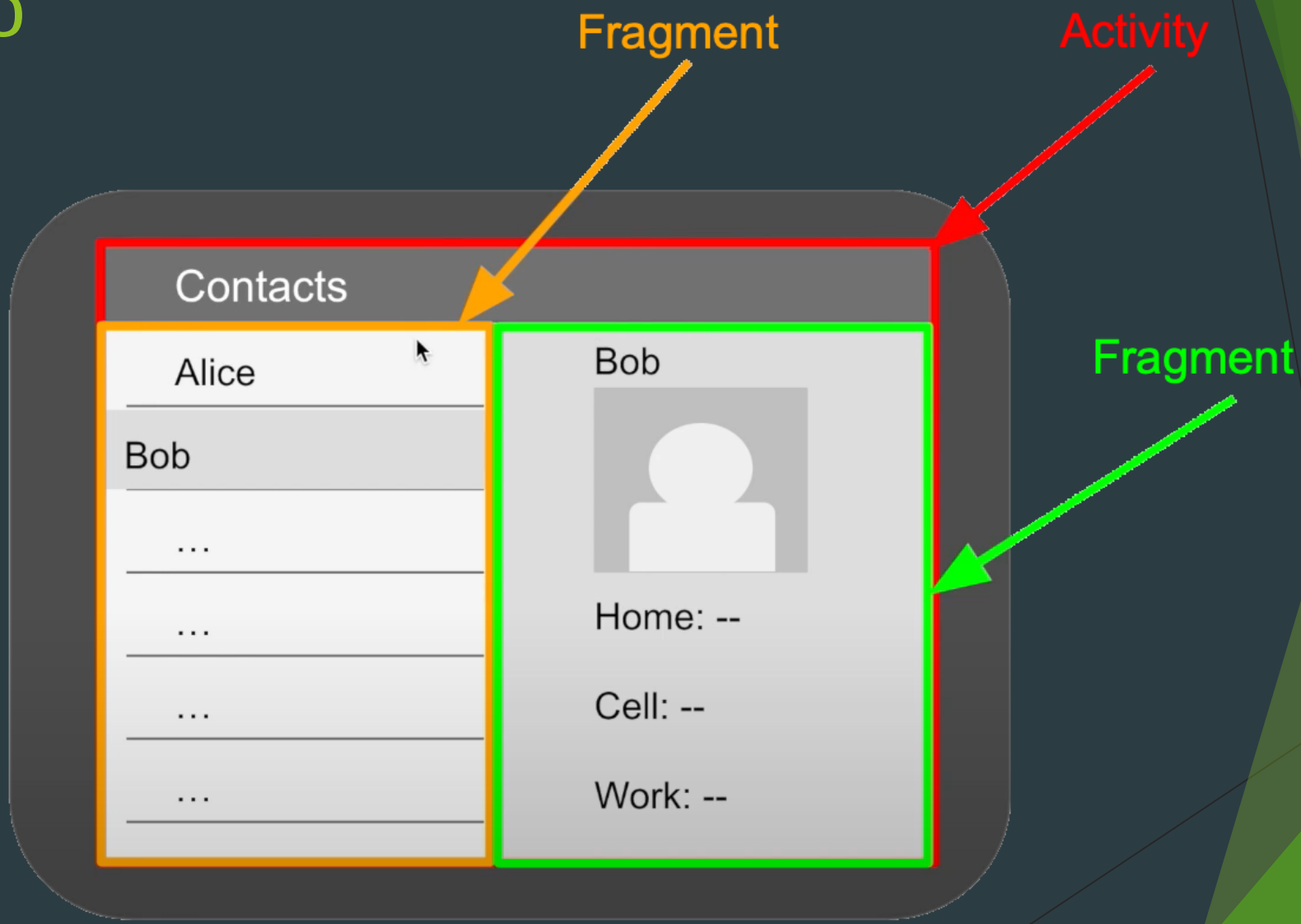
Esempio



Esempio



Esempio

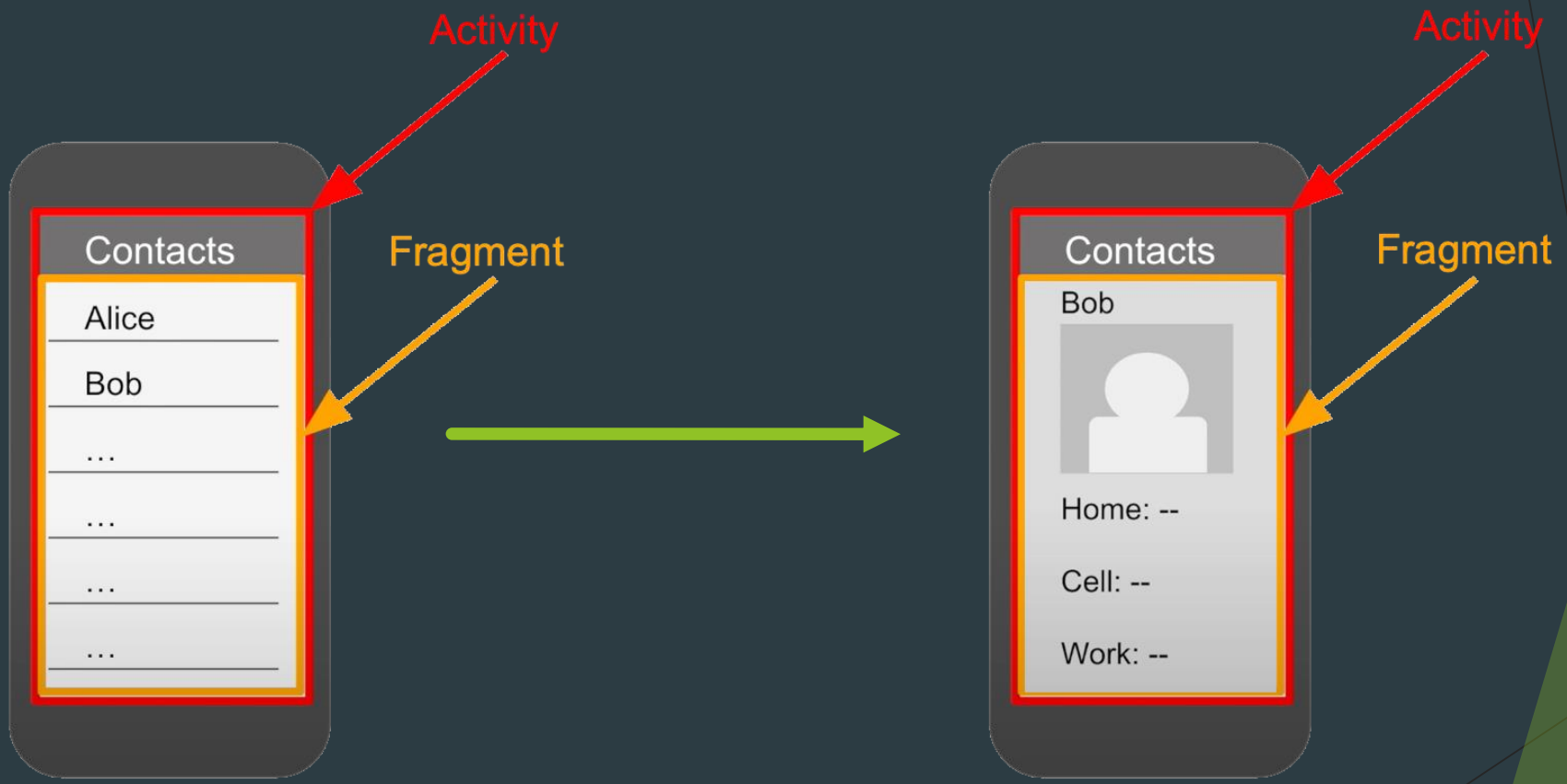

Fragment
Activity
Fragment
Contacts
Alice
Bob
…
…
…
…
Bob
Home: --
Cell: --
Work: --

Ancora sui fragment

- Come abbiamo già visto:
- i fragment sono una porzione della User Interface all'interno di un'activity
  - un'activity può contenere più fragments
  - è possibile utilizzare più volte lo stesso fragment in diverse activity

- La riusabili ci torntà dei fragment è utile soprattuto quando dobbiamo passare da un dispositivo ad un altro di diverse dimensioni
- Possiamo quindi creare una UI per uno smartphone in cui possiamo visualizzare in sequenza dei fragment che invece su un tablet vediamo sullo stesso layout

Esempio


Activity
Contacts
Fragment
Alice
Bob
…
…
…
…
Activity
Contacts
Fragment
Bob
Home: --
Cell: --
Work: --

Aggiungere fragment a runtime

- La classe FragmentManager fornisce metodi che consentono di aggiungere, rimuovere e sostituire fragment ad un'activity in fase di esecuzione al fine di creare un'esperienza dinamica
- È possibile aggiungere dei frament all'activity durante il runtime dell'attività. invece di definirli nel file di layout.
  - Ciò è necessario se si prevede di cambiare fragment durante la vita dell’activity
- Tramite FragmentManager si crea una FragmentTransaction, che fornisce API per aggiungere, rimuovere, sostituire ed eseguire altre transazioni
- Se l'attività consente di rimuovere e sostituire i fragment, è necessario aggiungere quelli iniziali dell'activity durante il metodo onCrea*te()*.

Aggiungere fragment a runtime

- il layout dell'activity deve includere una *View* contenitore, in cui è possibile inserire il fragment
  - È una regola importante, specialmente quando si aggiungono i fragment in fase di runtime.
- Per sostituire un fragment con un altro, il layout include un ***FrameLayout*** vuoto che funge da contenitore
- Per ottenere un ***FragmentManager*** bisogna chiamare ***getSupportFragmentManager()***
- Utilizzando ***beginTransaction()*** possiamo creare un ***FragmentTransaction*** e aggiungere un frammento con ***add()***
- È possibile eseguire più transazioni utilizzando la stessa ***FragmentTransaction***. Al temine delle modifiche basta chiamare ***commit()***

Esempio

- Si controlla se l'activity sta usando la versione del layout con frame_container come FrameLayout.
- Se è stata ripristinata da un precedente stato, allora non c'è bisogno di fare niente.
  - Bisognerebbe ritornare, oppure si può terminare con un overlapping fragment.
- Si crea il fragment da inserire nell'activity
- Nel caso in cui questa activity fosse stata iniziata con istruzioni speciali da una Intent, allora si passano gli extra dell'Intent al fragment come argomenti.
- Ora si può aggiungere il fragment al «fragment_container» FrameLayout.

```
public class MainActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState?) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.news_articles);


            if (findViewById(R.id.fragment_container) != null) {

            if (savedInstanceState != null) {
                return;
            }

            HeadlinesFragment firstFragment = new HeadlinesFragment();


            firstFragment.setArguments(getIntent().getExtras());

            getSupportFragmentManager().beginTransaction()
                    .add(R.id.fragment_container, firstFragment).commit();
        }
    }
}
```

Sostituire fragment

- Per sostituire un fragment si utilizza il metodo ***replace()***, nello stesso modo nel quale si utilizza add().
- Dopo transazioni di fragment, è spesso opportuno consentire all'utente di spostarsi indietro e "annullare" la modifica
  - per consentire all'utente di tornare indietro tra le transazioni è necessario chiamare ***addToBackStack()*** prima di eseguire il commit di FragmentTransaction
  - ***addToBackStack()*** accetta un parametro *stringa* facoltativo che specifica un nome univoco per la transazione, ma, a meno che non si pianifichi di eseguire operazioni avanzate, non è necessario

Esempio

```
//Crea un fragment e passagli come argomento l'oggetto che dovrebbe mostrare
Create fragment and give it an argument specifying the article it should show
ArticleFragment newFragment = new ArticleFragment();
Bundle args = new Bundle();
args.putInt(ArticleFragment.ARG_POSITION, position);
newFragment.setArguments(args);

FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();

//Sostituisci qualsiasi cosa ci sia nella view fragment_container con il nuovo fragment.
transaction.replace(R.id.fragment_container, newFragment);

// Aggiungi la transazione al back stack, così l'usr può navigare all'indietro
transaction.addToBackStack(null);

// Fai il commit  della transaction
transaction.commit();
```

Passaggio Dati tra Fragments

Introduzione

- A partire dalla versione 1.3.0-alpha04, ogni FragmentManager implementa FragmentResultOwner: ciò significa che un FragmentManager può agire da archivio centrale per il risultato di un fragment. Questo cambiamento permette a fragments separati di comunicare tra loro impostando e ascoltando i risultati senza che siano richiesti riferimenti diretti tra i fragments stessi.

Perché usare il Fragment Manager?

- Uno degli obiettivi più importanti dei fragments è quello di essere completamente indipendenti l'uno dall'altro. In presenza di layout variabili, c'è un'alta percentuale di errori causati dall'indisponibilità di alcuni fragments in alcuni scenari.
- Un Fragment Manager fornisce una soluzione affidabile per evitare queste situazioni.

CASO 1: Fragments Separati

- Passaggio dati da B verso A.

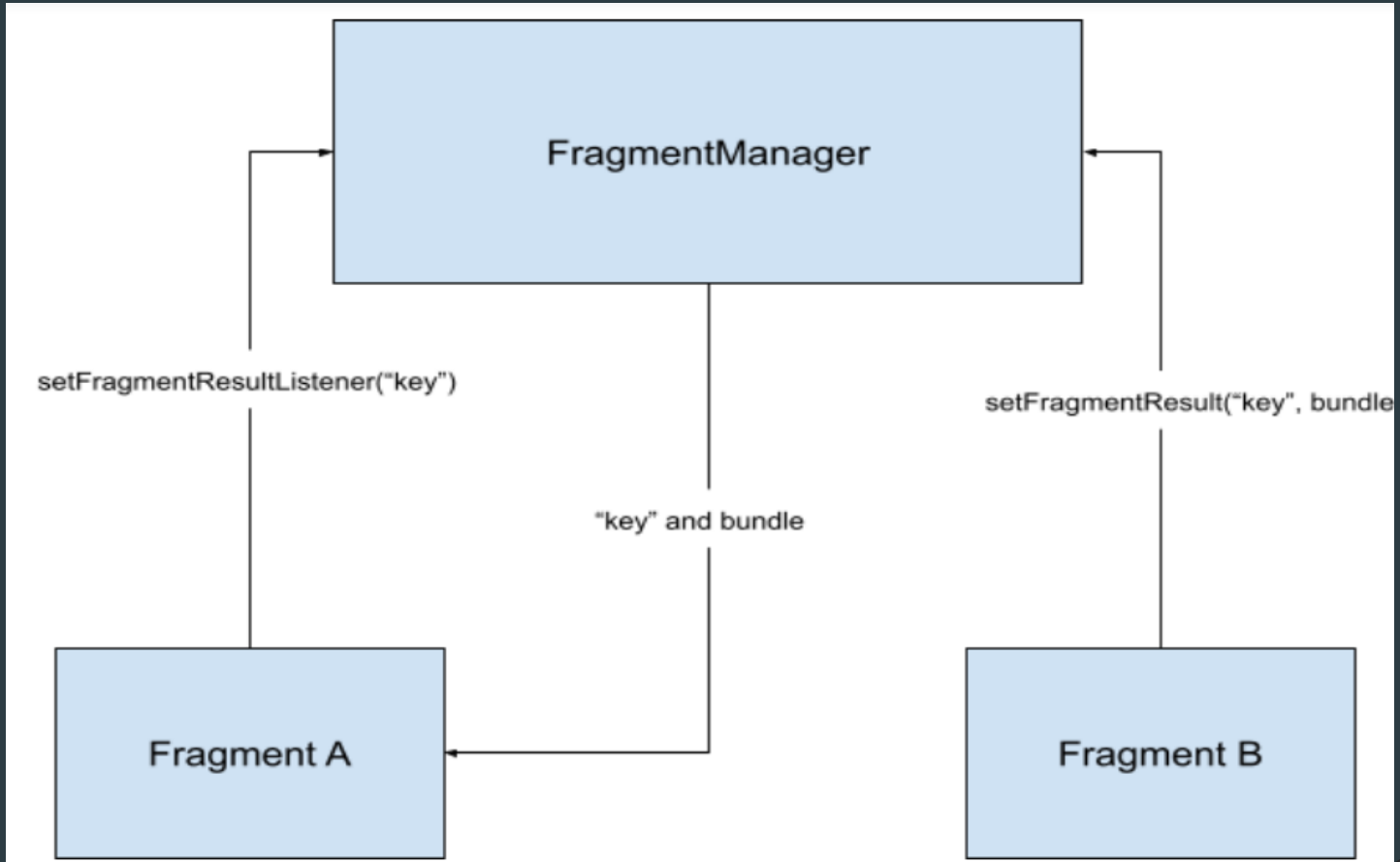


Figure 1. Fragment B sends data to fragment A using a `FragmentManager`.

CASO 1: Step 1

- Impostare un result listener sul fragment A.
- Chiamare la setFragmentResultListener() API sul Fragment Manager di A, come mostrato nell'esempio.

```
@Override
public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    getParentFragmentManager().setFragmentResultListener("key", this, new FragmentResultListener()
        @Override
        public void onFragmentResult(@NonNull String key, @NonNull Bundle bundle) {
            // We use a String here, but any type that can be put in a Bundle is supported
            String result = bundle.getString("bundleKey");
            // Do something with the result...
        }
    });
}
```

CASO 1: Step 2

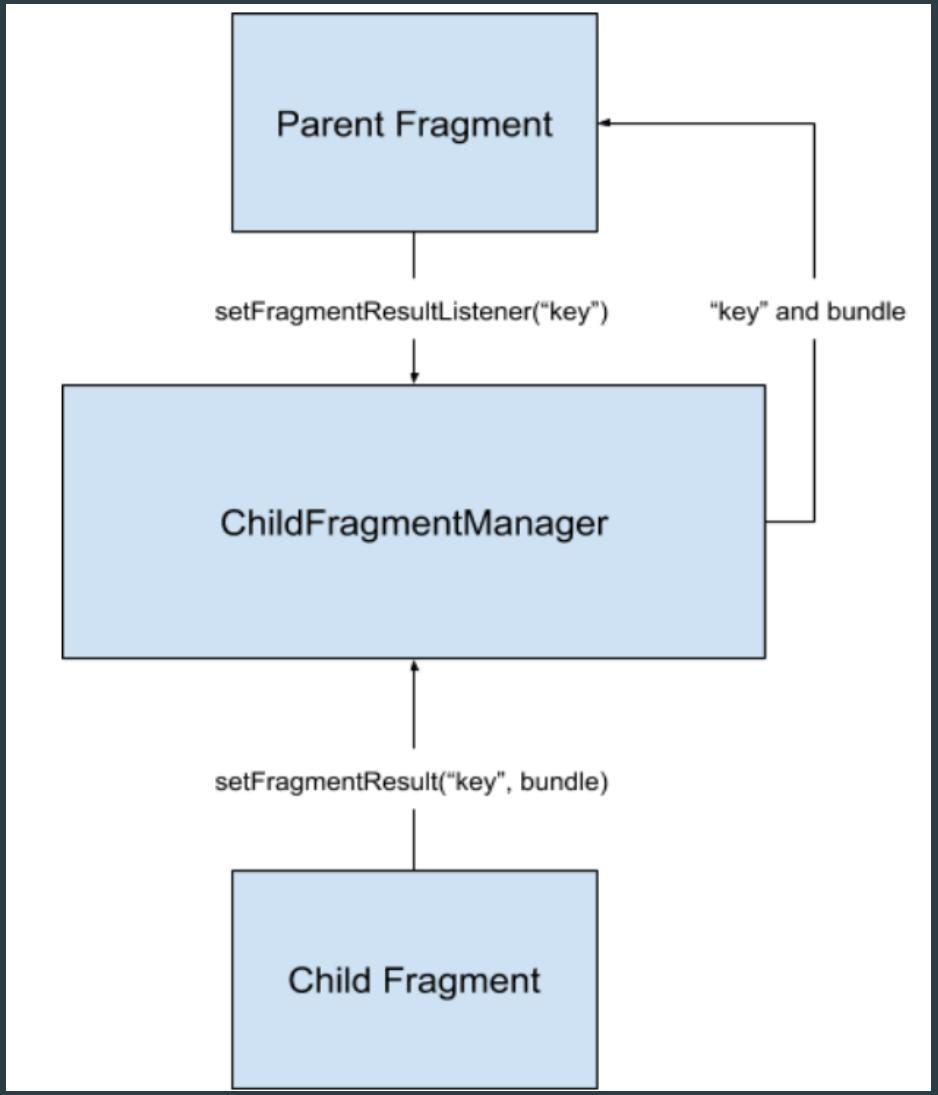
- Nel Fragment B, impostare il risultato nello stesso Fragment Manager,usando la stessa requestKey, utilizzando la setFragmentResult().
- Fragment A riceve il risultato ed esegue la listener callback.

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Bundle result = new Bundle();
        result.putString("bundleKey", "result");
        getParentFragmentManager().setFragmentResult("requestKey", result);
    }
});
```

Alcune Osservazioni

- Si può avere un solo listener e result per una determinata chiave.
- Se si chiama la setResult() più di una volta per la stessa chiave, il sistema invia al Fragment A il risultato più recente del Fragment B.
- Se si imposta un risultato senza un listener corrispondente per riceverlo, il risultato viene memorizzato nel FragmentManager fino ad impostare un listener con la stessa chiave.
- Si noti che il listener deve essere STARTED prima che il fragment possa ricevere il risultato. Una volta che un listener riceve un risultato e attiva la callback onFragmentResult(), il risultato viene eliminato. Questo comportamento ha due implicazioni principali:
  - I fragments per ricevere risultati devono essere nello stato STARTED.
  - Se un listener è avviato quando un risultato è già pronto, la callback è attivata immediatamente.

CASO 2: Fragments Padre E Figlio



CASO 2: STEP 1

- Il parent deve usare la getChildFragmentManager() quando chiama la setFragmentResultListener().

```
@Override
public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // We set the listener on the child fragmentManager
    getChildFragmentManager().setFragmentResultListener("key", this, new FragmentResultListener()
        @Override
        public void onFragmentResult(@NonNull String key, @NonNull Bundle bundle) {
            String result = bundle.getString("bundleKey");
            // Do something with the result..
        }
    });
}
```

CASO 2: STEP 2

- Il child passa il risultato tramite il suo fragment manager. Il parent riceverà il risultato quando il fragment sarà STARTED.

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Bundle result = new Bundle();
        result.putString("bundleKey", "result");

        // The child fragment needs to still set the result on its parent fragment manager
        getParentFragmentManager().setFragmentResult("requestKey", result);
    }
});
```

Comunicazione tra Fragments

Comunicazione tra due Fragment

- Spesso si vuole che due Fragment comunichino fra di loro, ad esempio in caso si volesse cambiare il contenuto in base ad un evento causato dall'utente
- La comunicazione diretta tra Fragment è pero sconsigliata, perché potrebbero verificarsi alcuni problemi
- La comunicazione deve passare tramite una ViewModel oppure tramite l'Activity associata ai Fragment.

Comunicazione tra due Fragment

- La strada raccomandata per la comunicazione tra Fragment è quella della creazione di una ViewModel.
- Gli oggetti ViewModel vengono conservati durante le modifiche di configurazione.
- Così le ViewModel consentono ai dati di essere conservati anche in caso di rotazione dello schermo.
- I Fragment possono aggiornare il dato con la ViewModel e se il dato è esposto usando LiveData, il nuovo stato può essere inserito per tutto il tempo in cui il Fragment osserva il LiveData dalla ViewModel

Comunicazione tra due Fragment

- Un altro modo per far comunicare due Fragment è quello di implementare manualmente un flusso di comunicazione usando le interfacce
- Questo metodo richiede però più lavoro per essere implementato e non è facilmente riusabile in altri Fragment

Comunicazione tramite ViewModel

- Come abbiamo detto i due Fragment condividono una ViewModel usando le loro Activity per gestire la comunicazione

```
public class SharedViewModel extends ViewModel {
    private final MutableLiveData<Item> selected = new MutableLiveData<Item>();

    public void select(Item item) {
        selected.setValue(item);
    }

    public LiveData<Item> getSelected() {
        return selected;
    }
}
```

- Il Fragment Master ottiene un' istanza di SharedViewModel
- Ad un certo evento dell'utente (un click), catturato da setOnClickListener(), verrà invocato il metodo select(item) che andrà a settare il valore del MutableLiveData

```
public class MasterFragment extends Fragment {
    private SharedViewModel model;

    public void onViewCreated(@NonNull View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        model = new ViewModelProvider(requireActivity()).get(SharedViewModel.class);
        itemSelector.setOnClickListener(item -> {
            model.select(item);
        });
    }
}
```

- Il DetailFragment crea a sua volta un'istanza di SharedViewModel e attende che il MutableLiveData sia settato dal Fragment Master
- A questo punto tramite il metodo geSelected() ottiene quel valore e può modificare la UI a suo piacimento

```
public class DetailFragment extends Fragment {
    public void onViewCreated(@NonNull View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        SharedViewModel model = new ViewModelProvider(requireActivity()).get(SharedViewModel.class);
        model.getSelected().observe(getViewLifecycleOwner(), { item ->
            // Update the UI.
        });
    }
}
```

- Notiamo quindi che in questo modo entrambi i Fragment prendono la stessa istanza di SharedViewModel che è correlata a questa activity
- Si hanno tre benefici
  - L'Activity non deve fare niente o sapere niente riguardo questa comunicazione
  - I Fragment non hanno bisogno di conoscere nulla l'uno dell'altro oltre la SharedViewModel
    - Se un Fragment scompare, l'altro continua a funzionare come al solito
  - Ogni Fragment ha il proprio ciclo di vita, e non è affetto da quello dell'altro Fragment

Navigare tra i Fragments utilizzando le animazioni

Frameworks per le animazioni

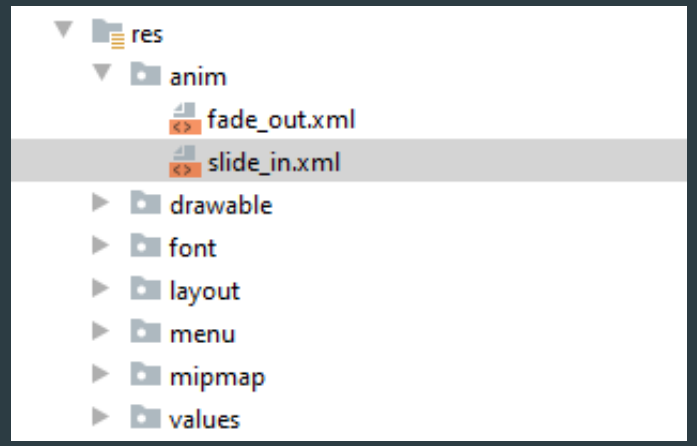
- La Fragment API offre due modi per utilizzare gli effetti di movimento e le trasformazioni per connettere visivamente i diversi fragment durante la navigazione.
- Un modo è l'utilizzo di Animation Framework, che utilizza le classi Animation e Animator.
- Un altro modo prevede l'utilizzo del Transition Framework, che include le transizioni di elementi condivisi.
- I due frameworks sono mutuamente esclusivi!

Frameworks per le animazioni

- Per ogni fragment è possibile specificare effetti personalizzati per l'ingresso e l'uscita dal fragment stesso e per le transizioni di elementi condivisi tra i fragment:
  - Un effetto di «entrata» determina come un fragment entra nello schermo, ad esempio il fragment potrebbe scorrere dal bordo dello schermo quando si naviga su esso.
  - Un effetto di «uscita» determina come un fragment esce dallo schermo, ad esempio si potrebbe dissolvere il fragment quando si naviga via da esso.
  - Una «transizione di elementi condivisi» determina il modo in cui una view condivisa tra due fragment si muove tra di essi.

Impostare un’animazione

- Innanzitutto, è necessario creare animazioni per gli effetti di ingresso e di uscita che vengono eseguite durante la navigazione verso un nuovo fragment.
- Queste animazioni possono essere definite nella directory res/anim:



Impostare un’animazione

- Supponiamo che si voglia realizzare un effetto in cui il fragment corrente scompaia e il fragment entrante scivoli dal bordo destro dello schermo:

fade_out.xml

```xml
<?xml version="1.0" encoding="utf-8"?>

<alpha xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="@android:integer/config_shortAnimTime"
    android:interpolator="@android:anim/decelerate_interpolator"
    android:fromAlpha="1"
    android:toAlpha="0" />
```

slide_in.xml

```xml
<?xml version="1.0" encoding="utf-8"?>

<translate xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="@android:integer/config_shortAnimTime"
    android:interpolator="@android:anim/decelerate_interpolator"
    android:fromXDelta="100%"
    android:toXDelta="0%" />
```

Utilizzare le animazioni

- Una volta definite le animazioni, si utilizzano chiamando FragmentTransaction.setCustomAnimations() passando le risorse di animazione tramite il loro resource ID, come mostrato in figura.

```
getSupportFragmentManager() FragmentManager
    .beginTransaction() FragmentTransaction
    .setCustomAnimations(
            R.anim.slide_in, // enter
            R.anim.fade_out  // exit
    ) FragmentTransaction
    .replace(R.id.listFrame,
        fragmentDetail) FragmentTransaction
    .addToBackStack("MASTER") FragmentTransaction
    .commit();
```

Effetti mediante Transition Framework

- Gli effetti di «entrata» e «uscita» possono essere definiti anche mediante il framework Transitions.
- L'utilizzo di questo framework è consigliato per la realizzazione di effetti che coinvolgono più di un tipo di animazione!
- Le transizioni possono essere definite nei file di risorse XML.

fade.xml

```
1  <!-- res/transition/fade.xml -->
2  <fade xmlns:android="http://schemas.android.com/apk/res/android"
3      android:duration="@android:integer/config_longAnimTime" />
```

slide_right.xml

```
1  <!-- res/transition/slide_right.xml -->
2  <slide xmlns:android="http://schemas.android.com/apk/res/android"
3      android:duration="@android:integer/config_longAnimTime"
4      android:slideEdge="right" />
```

Effetti mediante Transition Framework

- Dopo aver definito le transizioni, si applicano chiamando setEnterTransition() sul fragment in entrata e setExitTransition() sul fragment in uscita.

```
public class FragmentList extends Fragment implements SelectMode {

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_list, container, attachToRoot: false);
    TransitionInflater tranInflater = TransitionInflater.from(requireContext());
    setExitTransition(tranInflater.inflateTransition(R.transition.fade));
    return view;
  }
```

```
public class FragmentDetail extends Fragment {

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_detail, container, attachToRoot: false);
    TransitionInflater tranInflater = TransitionInflater.from(requireContext());
    setExitTransition(tranInflater.inflateTransition(R.transition.slide_right));
    return view;
  }
```

Test TheMealDB!