# Fragment e Navigation

## Fragment

Abbiamo visto come navigare tra activities, ma solitamente un’app non ha bisogno di diverse activity per ogni schermata proposta all’utente. A questo servono i **Fragments**

Un fragment è un pezzo di UI utilizzabile da una o più activities.

Text, application

Description automatically generated

In questa schermata, cliccare una delle sezioni del menu orizzontale non cambia activity, bensi’ ne modifica il contenuto alternando tra Fragments, mantenendo il menu, il FAB e il campo di ricerca nell’activity.

Il lifecycle del Fragment è simile a quello dell’Activity, ma leggermente piu’ complesso. I fragment hanno lifecycle diversi per il fragment stesso e la view (layout) la lui contenuta, in maniera tale da ottimizzare le risorse occupate quando ci si trova ad avere numerosi Fragment in memoria.

Graphical user interface, diagram

Description automatically generated

Una delle differenze fondamentali è la fase di creazione: mentre nell’activity utilizzeremmo onCreate per effettuare l’inflate ed il binding del layout, nel Fragment è necessario effettuare l’inflate in onCreateView, ed il binding in onViewCreated.

### Setup del progetto

Aggiungiamo il Navigation Component e il plugin Safe Args che ci aiuterà nello svilppo.

Nel *build.gradle:project*, aggiungi prima di *plugins* :

buildscript **{** ext **{** nav\_version = "2.5.3"  
 **}** dependencies **{** classpath("androidx.navigation:navigation-safe-args-gradle-plugin:$nav\_version")  
 **}  
}**

Aggiungi queste dipendenze in *build.gradle:app* :

implementation "androidx.navigation:navigation-fragment-ktx:$nav\_version"  
implementation "androidx.navigation:navigation-ui-ktx:$nav\_version"

Sempre nello stesso file, vai all’inizio. Nel tag *plugins*, aggiungi:

id 'androidx.navigation.safeargs.kotlin'

### Da Activity a Fragment

Trasforma l’esercizio lista da Activity a Fragment

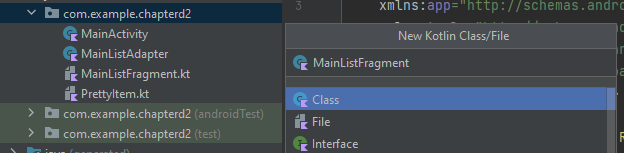
Per farlo dovremo portare la UI in un fragment, e trasformare la Main Activity in un contenitore di frammenti.

Comincia copiando il layout della main activity in un nuovo layout il cui nome comincia per **fragment\_**  
Ctrl-C –> Ctrl-V nell’albero files:

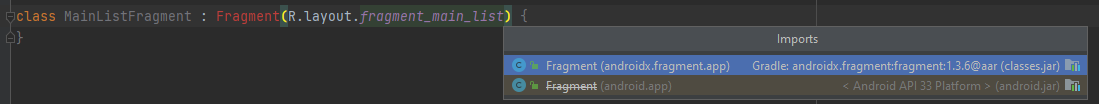
Graphical user interface, text, application

Description automatically generated

Ora crea un nuovo Fragment. Dovrai farlo a mano in quanto, alla data odierna, i modelli di Android Studio non sono aggiornati con le ultime versioni di Fragment.



Questa classe estenderà **Fragment(R.layout),** alla quale passeremo il layout appena creato



Questa versione di Fragment effettua automaticamente l’inflate, quindi non sarà per ora necessario implementare *onCreateView.*

Passiamo al binding facendo l’override di *onViewCreated*. Non dovendo piu’ effettuare l’inflate, utilizzeremo un metodo alternativo del viewBinding che permette di applicare un binding a un viewGroup:

Text

Description automatically generated

Questo è possibile perché, leggendo la doc, *view* è la view root ritornata dall’inflate onCreateView.

Qui copieremo il codice responsabile di popolare le view che avevamo scritto nella main activity.

Text

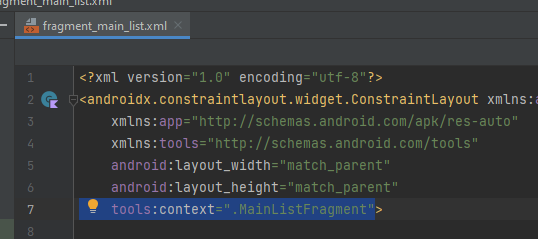
Description automatically generated

Copiamo anche il dataset, l’adapter e la funzione di generazione, e abbiamo completato la migrazione.

Text

Description automatically generated

Per semplificarci il lavoro e ottenere delle preview del fragment è necessario cambiare nel layout il parametro *tools:context* puntandolo al Fragment che abbiamo creato:



### Main Activity come contenitore di Fragments

Prima modifica MainActivity rimuovendo tutti i wiring e tenendo solo il binding:

Text

Description automatically generated

Poi modifica il layout di MainActivity :

* Rimuovi il FAB
* Cambia ConstraintLayout in FrameLayout
  + FrameLayout è un viewGroup molto leggero utilizzato spesso come parent di View singole.
* Sostituisci la RecyclerView con una  **FragmentContainerView**
  + <androidx.fragment.app.FragmentContainerView  
     android:id="@+id/nav\_host\_fragment"  
     android:name="androidx.navigation.fragment.NavHostFragment"  
     android:layout\_width="match\_parent"  
     android:layout\_height="match\_parent"  
     app:defaultNavHost="true"  
     app:navGraph="@navigation/nav\_graph"/>

Nel paragrafo successivo capiremo cos’è e come si crea il nav\_graph necessario a questa view.

Text

Description automatically generated

## Navigation Graph

I Fragment possono essere numerosi, e vengono utilizzati per avere una migliore gestione della navigazione tra le varie schermate dell’app. Ci sono alcuni modi per navigare tra i fragment, noi utilizzeremo il **Navigation Graph (per gli amici Nav Graph)**.

Il Navigation Graph è una rappresentazione schematica della navigazione tra Fragments.

Crea un nuovo file risorse di tipo Navigation, con nome *nav\_graph* – lo stesso impostato nel layout della Main Activity:  
Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Si aprirà un nuovo editor grafico, che vedrà a sinistra l’host *activity\_main* dove abbiamo impostato un riferimento a questo navgraph.

Aggiungi una nuova destinazione con il pulsante verde +, selezionando il Fragment che abbiamo creato

Graphical user interface, application

Description automatically generated

Ora il fragment verrà aggiunto al grafico, ed essendo il primo otterrà anche il flag *home* (l’icona della casa) che lo identifica come il primo del flusso.

Graphical user interface

Description automatically generated

## Navigazione tra frammenti

Vediamo ora come è possibile passare da un Fragment a un altro, rimanendo sempre all’interno della Main Activity.

### Layout e NavGraph

Crea una nuova classe Kotlin MainDetailFragment che useremo per mostrare il dettaglio di un oggetto della lista, poi utilizza l’assistente dell’IDE per creare un nuovo layout con il nome fragment\_main\_detail

Text

Description automatically generated

Copia il seguente layout:

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:padding="4dp"  
 tools:background="@color/black"  
 tools:context=".MainDetailFragment">  
  
 <ImageView  
 android:id="@+id/detail\_image\_iv"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:contentDescription="@string/image"  
 app:layout\_constraintBottom\_toTopOf="@id/detail\_id\_tv"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 tools:src="@android:drawable/gallery\_thumb" />  
  
 <TextView  
 android:id="@+id/detail\_id\_tv"  
 style="@style/TextAppearance.AppCompat.Large"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:background="@color/white"  
 android:textAlignment="center"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 tools:text="ID: 254" />  
  
</androidx.constraintlayout.widget.ConstraintLayout>

A picture containing text, monitor, screen, screenshot

Description automatically generated

Aggiungi questo Fragment nel nav\_graph, alla destra del fragment precedente

Graphical user interface

Description automatically generated

Ora trascina il lato destro di mainListFragment dentro mainDetailFragment: questo creerà una fraccia, chiamata **Action**

Diagram

Description automatically generated with medium confidence

Graphical user interface, application, Teams

Description automatically generated

Useremo questa action come destinazione per la nostra navigazione.

### Adapter

Ora introduciamo nell’adapter della lista la possibilità di cliccare un singolo elemento

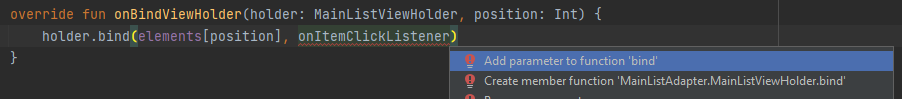
Andiamo a modificare il costruttore di MainListAdapter per includere una funzione Listener che comunicherà al fragment quando l’utente clicca sulla riga:

Graphical user interface, text, website

Description automatically generated

onItemClickListener è una funzione anonima, che prende in input un elemento PrettyItem e non ha output (Unit).

Aggiungiamo questo parametro alla funzione *bind* del ViewHolder:



Ora all’interno del binding, possiamo applicare un clickListener alla view root (l’intera riga) e inviare l’elemento che è stato cliccato a onItemClickListener

Graphical user interface, text

Description automatically generated

Ora quando una riga viene cliccata, *onItemClickListener* ci invierà l’oggetto cliccato.

Avendo modificato il costruttore di Adapter, dobbiamo anche cambiare come è invocato in MainListFragment, aggiungendo il nuovo clickListener con delle parentesi graffe:

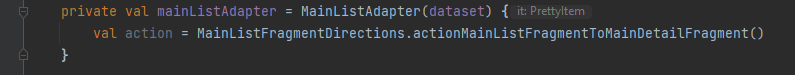
Text

Description automatically generated

### Implementazione in Fragment

Aggiungiamo la funzione di navigazione, che sarà avviata al click della riga.

In MainListFragment, nel listener dell’adapter, invoca questa classe:



Il plugin *SafeArgs* ha generato per noi i riferimenti all’action che abbiamo creato nel nav\_graph:

* *MainListFragmentDirections* : il nome del fragment di partenza + Directions
* *actionMyName* : il nome dell’action in formato *actionPartenzaToDestinazione*

Ora utilizziamo l’action come parametro per il NavController, una classe che si occupa della navigazione

Text

Description automatically generated

Cliccando una riga ora verrà visualizzato *MainDetailFragment*, ma sarà completamente bianco, in quanto non abbiamo legato ai campi nessun dato.

## Inviare dati tra frammenti

Ora inviamo a DetailFragment l’indirizzo dell’immagine, cosi’ da poter essere caricata nella ImageView.

Nel NavGraph, seleziona Detail Fragment cliccandoci. Nella colonna di destra, alla riga **arguments** creane di nuovi con il pulsante +

Graphical user interface, application

Description automatically generated

Crea due arguments:

1 – id, String  
2 - imageUrl, String

Poi fai una nuova build con Build -> Rebuild Project

Ora la funzione *actionStartToDestination* generata da SafeArgs richiederà questi due parametri.

Text

Description automatically generated

Dobbiamo ora passare questi parametri alla funzione. In precedenza, abbiamo creato il listener di MainListAdapter in modo che ritorni l’oggetto cliccato, quindi possiamo utilizzarlo per inviare i parametri al fragment che sarà mostrato al click.

Text

Description automatically generated

Ora bisogna recuperare i dati nel fragment di destinazione, generati per noi in una data class *FragmentNameArgs*. Useremo la funzione *by navArgs()*

****

* Questa è una *delegated property*: proprietà che vengono inizializzate solo una volta.  
  In particolare stiamo utilizzando l’inizializzazione *by lazy,* dove la variabile sarà inizializzata solo la prima volta che viene acceduta, utilizzando la funzione a destra di *by* .   
  <https://kotlinlang.org/docs/delegated-properties.html>

Questo oggetto contiene i parametri che abbiamo inviato dall’altro Fragment, e possiamo utilizzarlo per popolare il nostro layout:

Text

Description automatically generated

A picture containing text

Description automatically generated

## Setup Barra di Navigazione

È importante impostare la barra di navigazione presente in alto nella nostra app, perché non tutti i device hanno la possibilità di tornare indietro con il pulsante di sistema.

Questa barra è chiamata **Action Bar** e viene utilizzata per pulsanti che permettono azioni di navigazione o di setup.

La funzione per impostare la navigazione è attualmente buggata, utilizzeremo un workaround.

All'inizio di MainActivity aggiungi una variabile navController:

A screenshot of a computer

Description automatically generated with medium confidence

Nel binding:

A screen shot of a computer code

Description automatically generated with low confidence

* .post attende che navHostFragment abbia completato l’inflate
* Recupera e salva il navController tramite findNavController
* setupActionBarWithNavController informerà l’Action Bar di utilizzare come riferimento il navController

Nell’activity, effettua l’override della funzione onSupportNavigateUp, comunicando al sistema Android di utilizzare il nostro navController come riferimento.

A picture containing text, screenshot, font

Description automatically generated

Ora nella pagina di dettaglio sarà presente la freccia di navigazione.

A screenshot of a phone

Description automatically generated with low confidence

## Esercizio: Card Collection detail

* In Card Collection, dovrà essere possibile cliccare su una carta nella lista e navigare a una pagina di dettaglio dove vengono mostrati ID, valore e immagine.
* Aggiungi ad ogni elemento del dataset un campo “color” con un colore randomico
* Nel dettaglio, lo sfondo o la cornice dell’immagine deve essere del colore “color”
* Imposta correttamente la navigazione dell’Action Bar