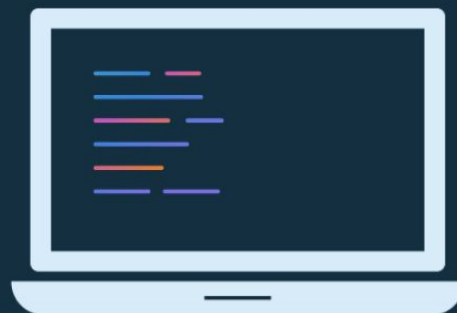


# Lezione 2.4

## App navigation



[Perpetuum mobile – Penguin Café Orchestra](#)



# Questa lezione

## Lezione 2.4: App navigation

- Activity multiple e intent
- App bar e Menu
- Fragments
- Navigazione in un'app
- Personalizzare la navigazione
- Interfaccia utente per la navigazione
- Task e back stack

# Activity multiple e intent

# Schermate multiple

Molto spesso le app forniscono funzionalità tramite diverse schermate

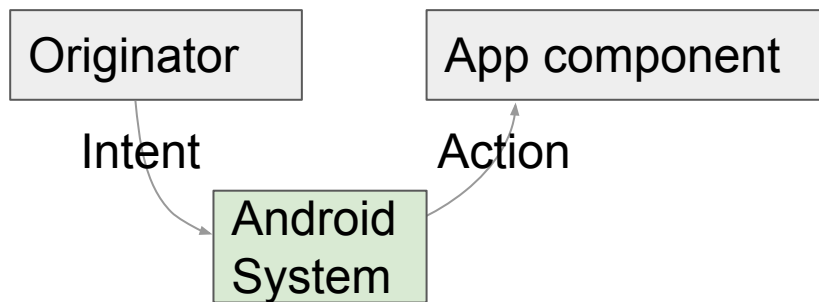
Esempi:

- Visualizzare i dettagli di un elemento (ad esempio, un prodotto specifico in un carrello della spesa)
- Creare un nuovo elemento (ad esempio, scrivere una mail)
- Pannello di configurazione di un'app
- Accesso a servizi di altre apps (ad esempio, Galleria di foto o Archivio di file)

# Cos'è un Intent?

Un Intent è una descrizione di un'operazione da svolgere

Si tratta di un oggetto usato per richiedere un'azione ad un'altra componente della stessa app, o di un'altra app, tramite il sistema operativo



# Cosa può fare un Intent?

- Lancia una Activity
  - Il click su un bottone lancia un'Activity per inserire del contenuto
  - Cliccando "Condividi" si apre un'app che ci permette di postare una foto
- Lancia un Service
  - L'app inizia a scaricare un file in background
- Invia un Broadcast
  - Il sistema informa tutte le app che il telefono è in carica

# Intent espliciti ed impliciti

## Explicit Intent

- Lancia una specifica Activity
  - Richiesta precisa come “Compra il té Earl Gray al negozio in piazza”
  - Esempio: MainActivity lancia ViewShoppingCartActivity

## Implicit Intent

- Chiede al sistema di trovare un'Activity che possa gestire una richiesta
  - Richiesta generica: “Cerca un negozio aperto che vende té nero>>
  - Esempio: cliccando “Condividi” si apre un pannello per scegliere da una lista di app

# Intent espliciti ed impliciti

- Un `Intent` implicito tipicamente richiede due informazioni:
  - L'azione da eseguire (ad esempio, `ACTION_VIEW`, `ACTION_EDIT`, `ACTION_MAIN`)
  - I dati su cui operare (ad esempio, i dettagli di un contatto)
- Uno esplicito indica invece la classe `Activity` da lanciare, ed eventualmente i dati su cui operare
- In questo modo si specifica la richiesta di transitare ad un'altra `Activity` (esplicita o meno) per compiere quell'azione con quei dati

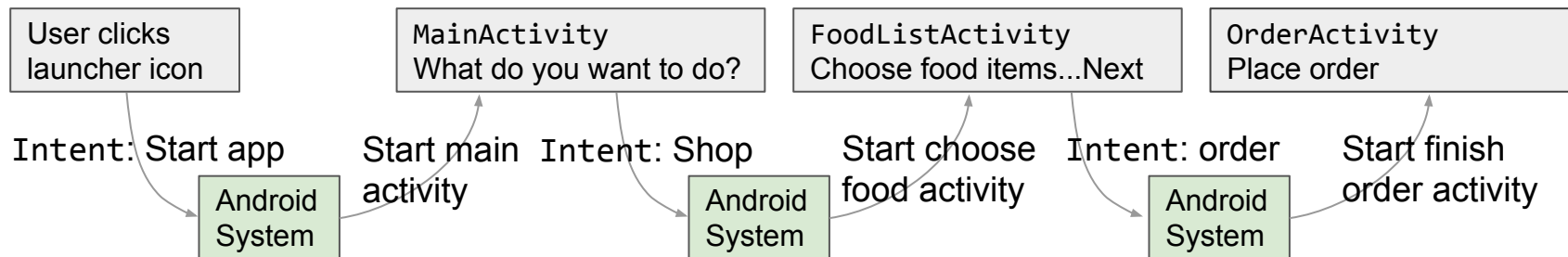


# Starting Activities



# Come vengono eseguite le Activity

- Tutte le istanze di Activity sono gestite dall'Android runtime
- Vengono lanciate con un "Intent" (un messaggio all'Android runtime che ne richiede l'esecuzione)



# Explicit intent

- Esaudisce la richiesta usando uno **specifico componente**
  - naviga internamente ad un'altra Activity della stessa app
  - naviga ad una specifica app esterna o un'altra app che hai sviluppato

# Lanciare un'Activity con un intent esplicito

Occorre usare un Intent esplicito

## 1. Creiamo l'Intent

- `intent = Intent(this, ActivityName::class.java)`

## 2. Usiamo l'Intent per lanciare l'Activity

- `startActivity(intent)`

Nota: il nome dell'Activity da lanciare è indicato come *riferimento* al tipo (alla classe). Per questo si utilizza la reflection ("::"). Se non la usassi, il compilatore penserebbe che sto cercando di istanziarla.

Nota 2: qui si fa riferimento alla classe di Runtime (.java)

# Implicit intent

- Fornisce una generica azione che l'app può eseguire
- Viene risolta mappando i tipi di dato coinvolti e l'azione a delle componenti esistenti nel device e note dal sistema operativo
- Consente ad ogni app che soddisfa i criteri di gestire la richiesta

# Lanciare un'Activity con un intent implicito

Usiamo un Intent implicito

## 1. Creiamo un'Intent

- `intent = Intent(action, uri)`

## 2. Usiamolo per lanciare l'Activity

- `startActivity(intent)`

# Intent impliciti : esempi

## Mostra una pagina web

```
val intent= Intent(Intent.ACTION_VIEW, Uri.parse("https://www.univpm.com/"))  
  
startActivity(intent)
```

Oppure

```
val intent = Intent(Intent.ACTION_VIEW)  
  
intent.setData(Uri.parse("https://www.univpm.com/"))  
  
startActivity(intent)
```

# Intent impliciti : esempi

## Componi un numero telefonico

```
val uri = Uri.parse("tel:8005551234")  
intent = Intent(Intent.ACTION_DIAL, uri)  
startActivity(it)
```



# Implicit intent

Lista dei più comuni tipi di intent impliciti:

<https://developer.android.com/guide/components/intents-common>

- Alarm clock
- Calendar
- Camera
- Contatti
- Email
- File storage
- Mappe
- Musica/video
- Note
- Telefono
- Ricerca
- Configurazione
- SMS
- Browser web

# Verificare che l'azione sia eseguibile

- Importante: verificare con `resolveActivity` che ci sia una componente capace di gestire l'azione
- Altrimenti, in mancanza di un riferimento, l'applicazione si chiuderà con un messaggio di errore
  - Nota: si tratta di un caso in cui non è possibile effettuare la verifica a compile-time

# Implicit intent: esempio

```
fun sendEmail() {  
    val intent = Intent(Intent.ACTION_SEND)  
    intent.type = "text/plain"  
    intent.putExtra(Intent.EXTRA_EMAIL, emailAddresses)  
    intent.putExtra(Intent.EXTRA_TEXT, "How are you?")  
  
    if (intent.resolveActivity(packageManager) != null) {  
        startActivity(intent)  
    }  
}
```

# Explicit intent: esempi

Navigazione tra activity della stessa app:

```
fun viewNoteDetail() {  
    val intent = Intent(this, NoteDetailActivity::class.java)  
    intent.putExtra(NOTE_ID, note.id)  
    startActivity(intent)  
}
```

Navigazione ad una specifica app esterna:

```
fun openExternalApp() {  
    val intent = Intent("com.example.workapp.FILE_OPEN")  
    if (intent.resolveActivity(packageManager) != null) {  
        startActivity(intent)  
    }  
}
```

# Inviare e ricevere dati

# Due tipi di approcci

- Data— una informazione rappresentabile come una URI
- Extras— una o più informazioni passate come una collection di chiavi-valori in un Bundle
  - Una map tra chiavi di tipo String e valori, impiegato nei messaggi
  - Si crea con `Bundle()`
  - Si aggiungono key-values con `putInt(key, value)`,  
`putDouble(key, value),...`
  - Si ottengono gli elementi con `getInt(key)`, `getDouble(key)...`

# Inviare e ricevere dati

Nell'Activity mittente:

1. Crea l'oggetto Intent
2. Inserisci dati o extras nell'Intent
3. Lancia la nuova Activity con `startActivity()`

Nell'Activity destinataria:

1. Ottieni un riferimento all'oggetto Intent con cui è stata lanciata
2. Recupera i dati o gli extras dall'Intent

# Usare un URI come dato

Una web page URL:

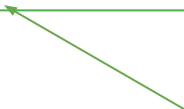
```
intent.setData(Uri.parse("http://www.google.com"))
```

Un esempio di file URI:

```
intent.setData(Uri.fromFile(File("/sdcard/sample.jpg")))
```



# Inserire informazioni negli extras

- `putExtra(name: String!, value: Double)`  
⇒ `intent.putExtra("level", 40.6)`
- `putExtra(name: String!, value: Int)`  
⇒ `intent.putExtra("age", 25)`
- `putExtras(bundle: Bundle)`  
⇒ se i dati sono molti, meglio creare un Bundle e passarlo
- Leggi la [documentazione](#) per tutte le opzioni

E' un metodo con molti overload, uno per ciascun tipo di dato

# Inviare dati ad un'Activity con extras

```
val intent = Intent(this, SecondActivity::class.java)
val message = "Hello Activity!"
intent.putExtra("messaggio", message)
startActivity(intent)
```

# Ricevere dati da un intent

- `getData()`

⇒ `val myUri = intent.getData()`

La variabile intent è automaticamente disponibile

- `getIntExtra (name: String, defaultValue: Int)`

⇒ `val level = intent.getIntExtra("myKey", 0)`

- `val bundle = intent.getExtras()`

⇒ Ottieni tutti i dati insieme come bundle

Se non esiste la chiave indicata viene fornito il defaultValue

# Ritornare i dati all'attività iniziale

1. Usa `startActivityForResult()` per lanciare la seconda Activity
2. Per ritornare i dati dalla seconda Activity:
  - Crea un nuovo Intent
  - Inserisci i dati nell'Intent usando `putExtra()`
  - Setta il risultato al valore `Activity.RESULT_OK` o `RESULT_CANCELED`
  - chiama `finish()` per chiudere l'Activity
3. Implementa `onActivityResult()` nella prima Activity

# startActivityResult()

startActivityResult(intent, requestCode)

- Il metodo lancia l'Activity (intent) e le assegna un identificatore (requestCode)
- I dati da ritornare vengono inseriti negli extras
- Quando ha terminato, viene effettuata un pop dallo stack delle Activity esistenti. Il sistema ritorna dunque alla precedente Activity ed esegue la funzione di callback onActivityResult() per processare i dati ritornati
- Il requestCode è un *token* che viene usato per identificare quale Activity ha ritornato i valori

# 1. startActivityForResult(): esempio

```
val CHOOSE_FOOD_REQUEST = 1 //requestCode  
val intent = Intent(this, ChooseFoodItemsActivity::class.java)  
startActivityForResult(intent, CHOOSE_FOOD_REQUEST)
```

## 2. Ritorna i dati e termina la seconda Activity

Creiamo l'intent

```
val replyIntent = Intent()
```

Inseriamo i dati da ritornare negli extras

```
replyIntent.putExtra("valore", 50);
```

Setta il risultato al valore `RESULT_OK`

```
setResult(RESULT_OK, replyIntent)
```

Termina l'activity corrente

```
finish()
```

### 3. Implementa onActivityResult()

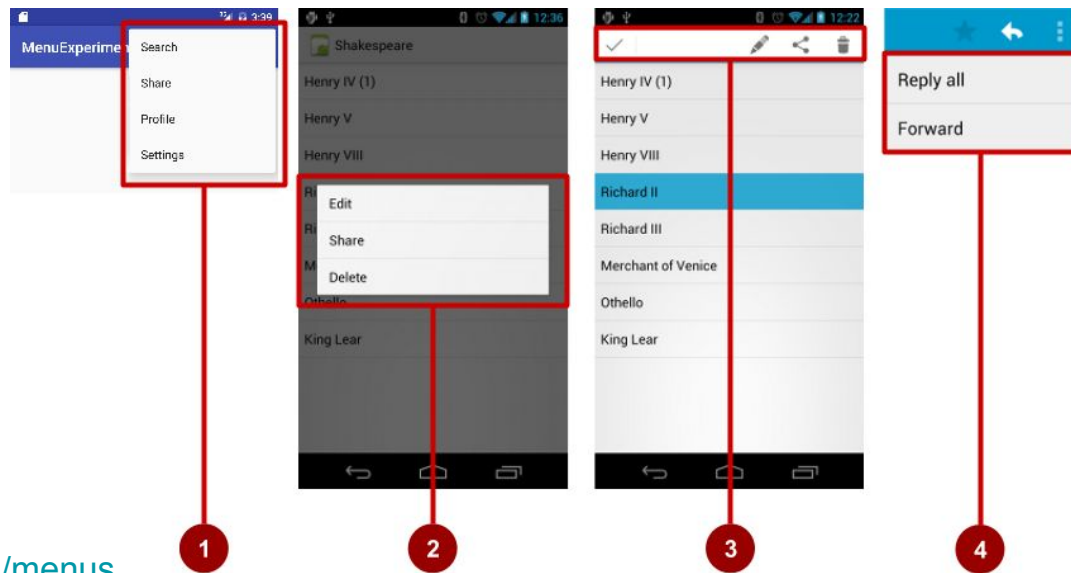
```
override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == CHOOSE_FOOD_REQUEST) // Identify activity
        if (resultCode == RESULT_OK) { // Activity succeeded
            val reply = data.getStringExtra("valore")
            // ... do something with the data
        }
    }
}
```



# Menu

# Types of Menus

1. App bar
2. Floating context menu
3. Contextual action bar
4. Popup menu



Docs: <https://developer.android.com/guide/topics/ui/menus>

# App bar

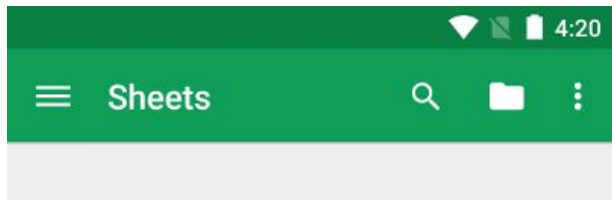
# App bar

L'app bar è un elemento centrale nello sviluppo di un'interfaccia.

- rende la UI consistente con le altre app Android
- Permette agli utenti di capire rapidamente come funziona l'app e migliora l'esperienza dell'utente

Ha tre funzioni chiave:

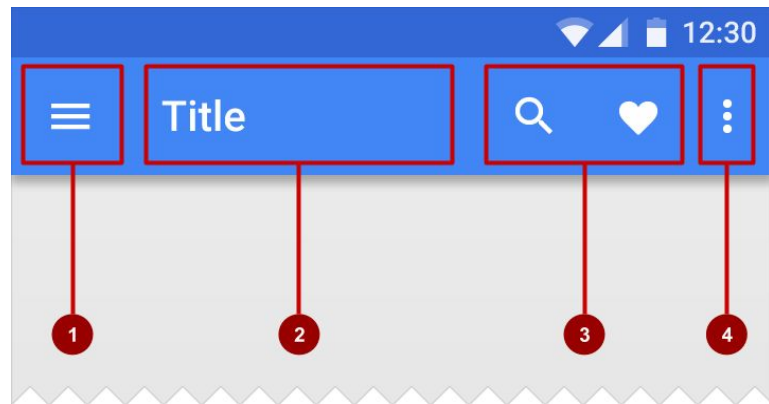
- Fornisce uno spazio per dare identità all'app ed indicare all'utente dove si trova nell'app
- Permette di accedere alle azioni principali in modo predicibile (ad esempio, la ricerca)
- Supporta la navigazione ed il cambio di view (tramite tabs o drop-down list)



# App bar

Componenti:

1. Nav icon per aprire il Navigation Drawer
2. Titolo dell'Activity corrente
3. Icone di opzioni di menù
4. *Action overflow button* per le restanti opzioni di menu



# App bar: ActionBar

A partire da Android 3.0 (API 11), tutte le activity usano **ActionBar** come barra superiore dell'app. Nelle versioni successive sono state aggiunte features specifiche per diverse release di Android, che funzionano in modo diverso in base alla versione di Android sul device.

# App bar: Actionbar (esempio)

Ogni schermata che non è l'Entry point dovrebbe fornire un meccanismo per tornare all'Activity di provenienza. Un modo di farlo è tramite l'App bar

- Per aggiungere un bottone “◀” occorre dichiarare nel file *AndroidManifest* quale Activity è il “logical parent” di quella secondaria

```
<activity android:name=".SecondActivity"
    android:parentActivityName=".MainActivity">
    <!-- The meta-data tag is required if you support API level 15 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
```

# App bar: Toolbar

Spesso però è preferibile usare come app bar una **Toolbar**, che garantisce un comportamento consistente tra le versioni di Android. Per farlo occorre indicare in AndroidManifest di usare un *theme* di riferimento senza la vecchia ActionBar:

```
<application
    android:theme="@style/Theme.AppCompat.Light.NoActionBar"
/>
```

Oppure, possiamo creare un nostro theme (come vedremo più avanti) indicando però, ad esempio:

```
<style name="AppTheme" parent="android:Theme.AppCompat.Light.NoActionBar"
    <item name=.../>
    ...
</style>
```



# Aggiungere un'App bar

Inserire la View corrispondente nel layout e posizionarla in cima, ad esempio:

```
<androidx.appcompat.widget.Toolbar
```

```
    android:id="@+id/my_toolbar"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="?attr/actionBarSize"
```

```
    android:background="?attr/colorPrimary"
```

```
    android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
```

```
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
```

```
    app:navigationIcon="?homeAsUpIndicator"
```

```
/>
```

Questo serve per inserire la freccia "up" e tornare all'Activity precedente. Non serve inserirlo nella prima Activity

# 1) Impostare l'App bar

In onCreate, impostare l'app bar, che di default visualizzerà il nome dell'app:

```
setSupportActionBar(findViewById<Toolbar>(R.id.my_toolbar))
```

L'inserimento di elementi nell'App bar si effettua creando un menu ed inserendo dei menu items. Il menù verrà poi associato tramite inflating alla Toolbar.

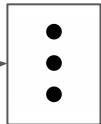
## 2) Creare un Menu

Definisci gli elementi del menu in una risorsa menu in XML (localizzata in res/menu)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
  <item
    android:id="@+id/action_settings"
    android:orderInCategory="100"
    android:title="@string/action_settings"
    app:showAsAction="never" />
</menu>
```

Definisce se mostrare l'item nella Toolbar (altrimenti viene raggruppato nel "overflow menu")

- ifRooms = se c'è spazio
- never = mai (cioè va sempre nel overflow menu)
- always = sempre visibile



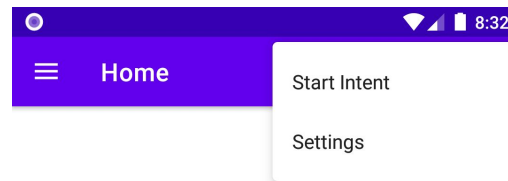
# Esempio di opzioni di menù

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/action_intent"
          android:title="@string/action_intent" />

    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never" />

</menu>
```



### 3) Aggiunta del menu all'Activity

```
override fun onCreateOptionsMenu(menu: Menu): Boolean {  
    menuInflater.inflate(R.menu.main, menu)  
    return true  
}
```

## 4) Gestione delle opzioni selezionate

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {  
    when (item.itemId) {  
        R.id.option1 -> {  
            // do something  
        }  
        R.id.option2 -> {  
            // do something else  
        }  
        else -> // do something else  
    }  
    ...  
}
```

## 4) Gestione delle opzioni selezionate

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {  
    when (item.itemId) {  
        R.id.option1 -> {  
            val intent = Intent(Intent.ACTION_WEB_SEARCH)  
            intent.putExtra(SearchManager.QUERY, "pizza")  
            if (intent.resolveActivity(packageManager) != null) {  
                startActivity(intent)  
            }  
        }  
        else -> Toast.makeText(this, item.title, Toast.LENGTH_LONG).show()  
    }  
    ...  
}
```

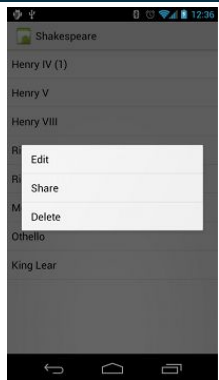
# Contextual menu & contextual action bar



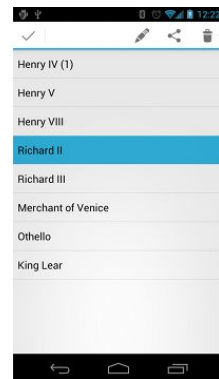
# I contextual menu

- Permettono agli utenti di eseguire azioni sulle View selezionate
- Possono essere agganciate ad ogni View (spesso utilizzate nei RecyclerView, GridView, o altre collezioni di View)

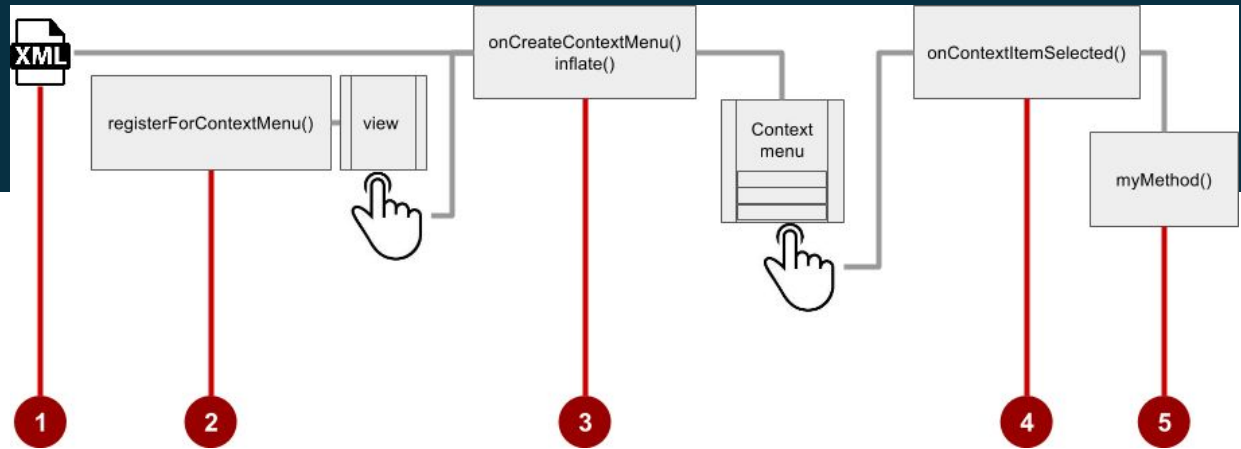
# I contextual menu



- Floating context menu—dopo un click lungo su una view
  - Per modificare la View o accedere a suoi dettagli (l'utente interagisce con una View alla volta)
- Contextual action mode—action bar temporanea al posto dell'app bar
  - L'azione si applica alle View selezionate (l'utente può interagire con più View alla volta)



# Steps



1. Crea un menu XML e definiscine l'aspetto e la posizione
2. Registra una View usando `registerForContextMenu()`
3. Implementa `onCreateContextMenu()` nell'Activity per inserire il menu
4. Implementa `onContextItemSelected()` per gestire i click sul menu item
5. Crea i metodi per gestire le azioni per ciascun elemento di menu

# Creazione del menu

## 1. Crea un menu XML (menu\_context.xml)

```
<item  
    android:id="@+id/context_edit"  
    android:title="Edit"  
    android:orderInCategory="10"/>
```

```
<item  
    android:id="@+id/context_share"  
    android:title="Share"  
    android:orderInCategory="20"/>
```

# Registra una view ad un context menu

2) Nella onCreate() dell'Activity registra una View per visualizzare un context menu:

```
val article_text = findViewById<TextView>(R.id.article)  
registerForContextMenu(article_text)
```

# Implementa onCreateContextMenu()

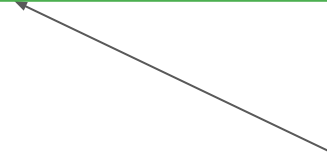
## 3. Specifica quale context menu:

```
override fun onCreateContextMenu(menu: ContextMenu, v: View,  
                                menuInfo: ContextMenu.ContextMenuInfo) {  
    super.onCreateContextMenu(menu, v, menuInfo)  
    val inflater: MenuInflater = menuInflater  
    inflater.inflate(R.menu.context_menu, menu)  
}
```

# Implementa onContextItemSelected()

```
override fun onContextItemSelected(item: MenuItem): Boolean {  
    val info = item.menuInfo as AdapterView.AdapterContextMenuInfo  
    return when (item.itemId) {  
        R.id.edit -> {  
            editNote(info.id)  
            true  
        }  
        R.id.delete -> {  
            deleteNote(info.id)  
            true  
        }  
        else -> super.onContextItemSelected(item)  
    }  
}
```

Questa variabile permette di accedere ad informazioni sul particolare elemento che ha generato il context menu



# Cos'è l'Action mode?

- Una modalità della UI che permette di sostituire parti della normale UI temporaneamente
- Ad esempio: un action mode potrebbe attivarsi selezionando una porzione di testo o premendo a lungo su un elemento



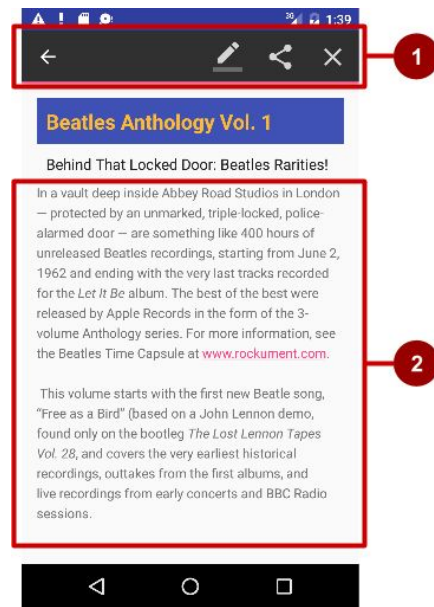
# L'Action mode ha un ciclo di vita

- Inizia con `startActionMode()` (ad esempio, in un `OnLongClickListener`)
- L'interfaccia `ActionMode.Callback` fornisce metodi per gestire il ciclo di vita che vanno sovrascritti:
  - `onCreateActionMode(ActionMode, Menu)` alla creazione iniziale
  - `onPrepareActionMode(ActionMode, Menu)` dopo la creazione e ogni volta che l'`ActionMode` viene invalidato
  - `onActionItemClicked(ActionMode, MenuItem)` ogni volta che un Contextual action button viene cliccato
  - `onDestroyActionMode(ActionMode)` quando l'Action mode viene chiusa

# Cos'è una contextual action bar?

La contextual action bar è una barra mostrata quando si preme a lungo su determinate Views:

1. Contextual action bar includono diverse azioni da compiere
  - Es: **Edit**, **Share**, e **Delete**
  - **Done** (l'icona con la freccia) a sinistra che fa scomparire l'Action bar
2. La View su cui un long click attiva l'Action bar

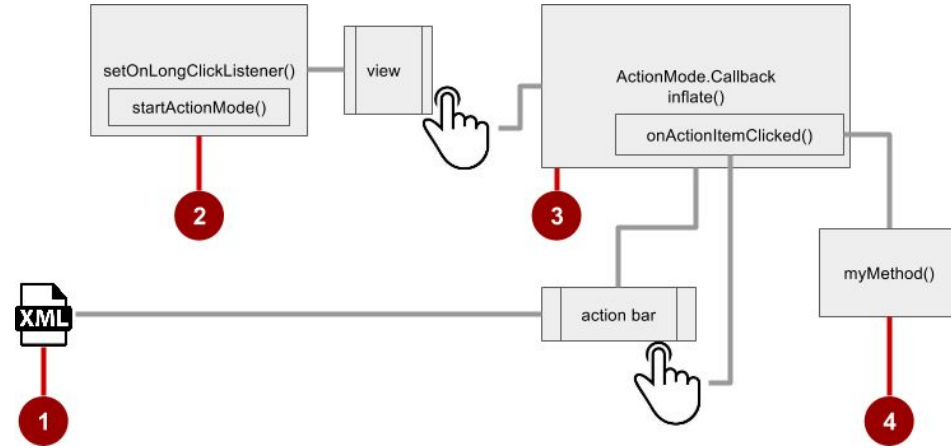


# Step

- 1) Creare un menu XML con le icone per gli item
- 2-3) Configurare l'Action Mode (due approcci: per la singola View o per una lista di elementi)
- 4) Creare metodi per eseguire le azioni previste dai click sugli item

# Step per una singola View

1. Creare un menu XML con le icone per gli item
2. Definire `setOnLongClickListener()` sulla View che attiva la bar e chiamare `startActionMode()` per gestire il click



3. Implementare l'interfaccia `ActionMode.Callback` per gestire il ciclo di vita dell'Action mode; includere azioni di risposta per i click sugli item del menu con `onActionItemClicked()`
4. Creare metodi per eseguire le azioni previste dai click sugli item

## 2) Uso di setOnLongClickListener

In onCreate():

```
someView.setOnLongClickListener { view ->
    // Called when the user long-clicks on someView
    when (actionMode) {
        null -> {
            // Start the CAB using the ActionMode.Callback defined above
            actionMode = activity?.startActionMode(actionModeCallback)
            view.isSelected = true
            true
        }
        else -> false
    }
}
```

### 3) Implementare ActionMode.Callback

```
private val actionModeCallback = object :  
    ActionMode.Callback {  
    // qui implemento i metodi necessari del punto 4  
}
```

# Step per una lista di elementi

1. Creare un menu XML con le icone per gli item
2. Inserire una ListView nel layout.  
Dichiarare un ArrayAdapter ed iniziarlo con un array di valori.  
Associare l'ArrayAdapter alla ListView
3. Settare la scelta multipla per la ListView. Associare un `MultiChoiceModeListener` alla ListView ed implementare l'interfaccia `AbsListView.MultiChoiceModeListener`
4. Creare al suo interno metodi per eseguire le azioni previste dai click sugli item

## 2-3) Setto l'adapter alla ListView

In onCreate():

```
val arrayAdapter: ArrayAdapter<String> =  
    ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, myArray)  
val myListView = findViewById<ListView>(R.id.myListView)  
myListView.adapter = arrayAdapter  
  
myListView.choiceMode = ListView.CHOICE_MODE_MULTIPLE_MODAL  
  
myListView.setMultiChoiceModeListener(object : AbsListView.MultiChoiceModeListener {  
  
    // qui implemento i metodi necessari del punto 4  
})
```



## 4) Implementare onCreateActionMode

```
override fun onCreateActionMode(mode: ActionMode, menu: Menu): Boolean {  
    // Inflate a menu resource providing context menu items  
    val inflater: MenuInflater = mode.menuInflater  
    inflater.inflate(R.menu.context_menu, menu)  
    return true  
}
```

## 4) Implementare onPrepareActionMode

- Chiamata ogni volta che l'action mode viene mostrata
- Viene sempre chiamata dopo onCreateActionMode, ma può essere chiamata più volte se l'action mode viene invalidata

```
override fun onPrepareActionMode(mode: ActionMode, menu: Menu):  
Boolean {  
    return false // Return false if nothing is done  
}
```

## 4) Implementare onOptionsItemSelected

- Chiamata quando gli utenti selezionano un azione
- I click vengono gestiti in questo modo

```
override fun onOptionsItemSelected(mode: ActionMode, item: MenuItem): Boolean {  
    return when (item.itemId) {  
        R.id.menu_share -> {  
            shareCurrentItem()  
            mode.finish() // Action picked, so close the CAB  
            true  
        }  
        else -> false  
    }  
}
```

## 4) Implementare onDestroyActionMode

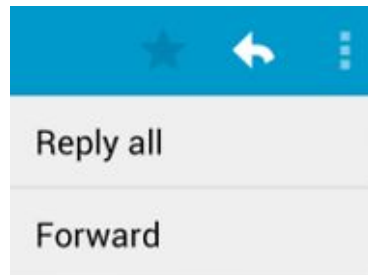
- Chiamata quando l'utente esce dall'action mode

```
override fun onDestroyActionMode(mode: ActionMode) {  
    actionMode = null  
}
```

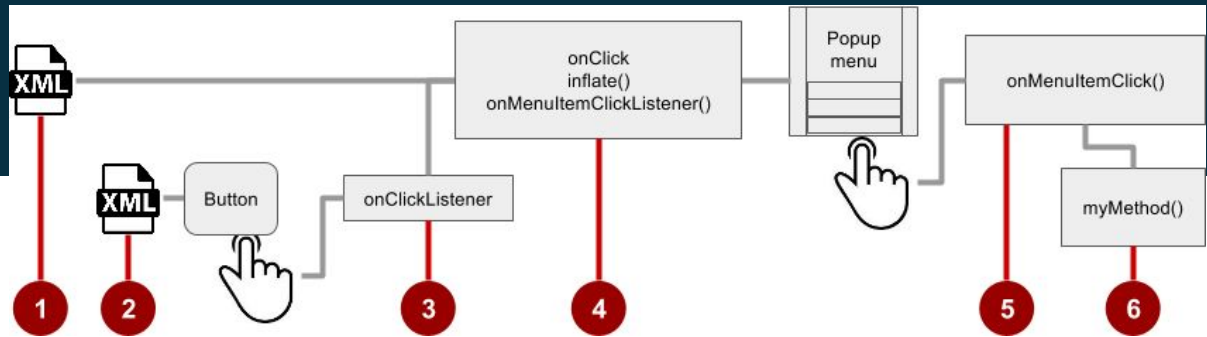
# Popup menu

# Cos'è un popup menu?

- Lista verticale di elementi ancorati ad una view (spesso ad un'icona)
- Le azioni non dovrebbero influenzare direttamente il contenuto della View
  - Es: le opzioni del menu overflow
  - Es: nelle app di mail, **Reply All** e **Forward** hanno a che fare con il messaggio ma non ne modificano il contenuto direttamente



# Steps



1. Crea un menu XML con le opzioni da visualizzare
2. Aggiungi un Button e assegnagli un `onClick` listener
3. Sovrascrivi `onClick()` per effettuare l'inflating del popup
4. Definisci le azioni da compiere sul click degli elementi con `onMenuItemClickListener()`

# Implementa onClick e onOptionsItemSelected

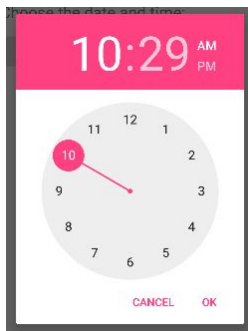
```
button.setOnClickListener {  
    val popupMenu = PopupMenu(this, button)  
    popup.menuInflater.inflate(R.menu.menu_popup, popup.menu)  
    popup.setOnMenuItemClickListener( { item ->  
        when (item.itemId) {  
            R.id.option1 -> /* do something */  
            R.id.option2 -> /* do something else */  
        }  
        true})  
    popupMenu.show()  
}
```



# Dialogs

# Dialogs

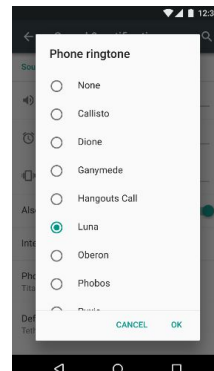
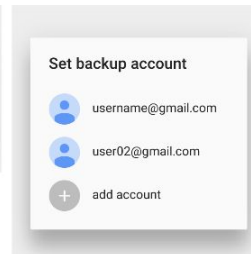
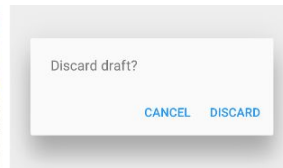
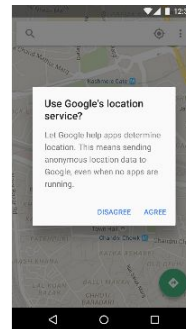
- I Dialog sono finestre modali: appaiono in primo piano, interrompendo il flusso delle Activity, e richiedono un'azione dell'utente per proseguire



TimePickerDialog



DatePickerDialog

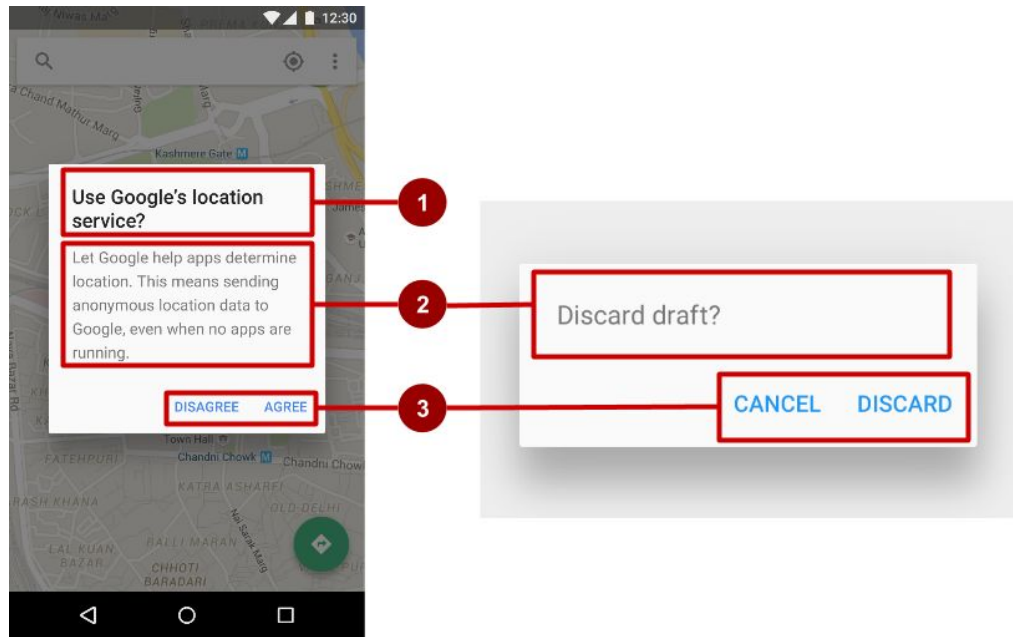


AlertDialog

# AlertDialog

AlertDialog può mostrare:

1. Un titolo (opzionale)
2. Una content area
3. Degli action buttons



# Costruire l'AlertDialog

Usiamo `AlertDialog.Builder` per costruire un alert dialog e settarne gli attributi

```
val alertDialog = AlertDialog.Builder(this)
alertDialog.setTitle("Connect to Provider")
alertDialog.setMessage("Message")
// ... Codice per definire le azioni

alertDialog.show()
```

# Definire le azioni per i bottoni

- `alertDialog.setPositiveButton()`
- `alertDialog.setNeutralButton()`
- `alertDialog.setNegativeButton()`

# Definire le azioni per i bottoni

```
AlertDialog.setPositiveButton(  
    "OK", DialogInterface.OnClickListener {  
        override fun onClick(dialog: DialogInterface, which: Int) {  
            // User clicked OK button.  
        }  
    })
```

Lo stesso pattern vale per `setNegativeButton()` e `setNeutralButton()`

# Fragments

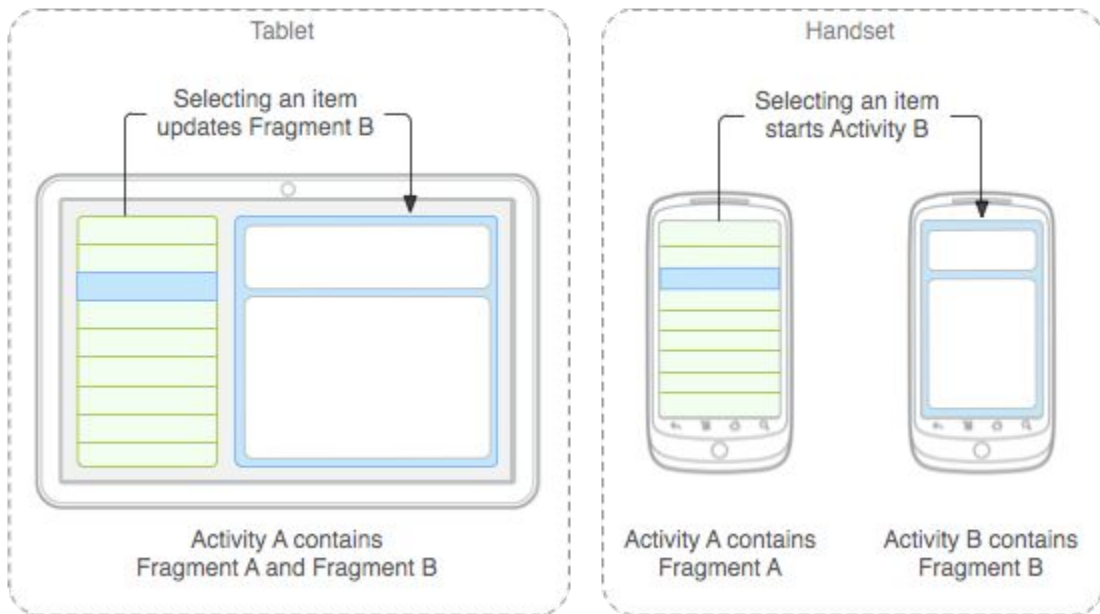
# Fragment

- Rappresenta un behavior o una porzione di UI in un'activity ("microactivity")
- Deve essere ospitato da un'activity, a cui è legato il suo ciclo di vita
- Possono venire aggiunti o rimossi a runtime
- Usa la versione AndroidX della classe `Fragment` (`androidx.fragment.app.Fragment`), invece di quella standard (`android.app.Fragment`), che è stata deprecata dalle API 28



# Fragments per diversi layout layouts

I fragment sono utili come elementi con cui comporre un'activity.



# Creare un Fragment

1. Creare un file .kt con una classe che estende `Fragment`
2. Sovrascrivere un metodo `onCreateView` ed effettuare l'inflate del layout:

```
override fun onCreateView(inflater: LayoutInflater, container:
ViewGroup?, savedInstanceState: Bundle?): View? {

    val v = inflater.inflate(R.layout.book_titles, container, false)

    return v

}
```

# Creare un Fragment

- Per semplicità si può anche scrivere:

```
class ExampleFragment : Fragment(R.layout.book_titles)
```

# Creare un Fragment

3. Sovrascrivere un metodo `onViewCreated`, che viene eseguito subito dopo la creazione dell'Fragment, dove effettuo l'inizializzazione degli elementi

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) { }
```

# Creare un Fragment

4. Associare il Fragment all'Activity. Si può fare in modo statico (a design-time) o dinamico (a run-time)

Approccio **statico**: si inserisce nel layout dell'Activity una View “fragment” con specificato il Fragment da visualizzare come valore dell'attributo `android:name`

# Creare un Fragment

Approccio **dinamico**: istanzio il fragment all'interno dell'activity e poi lancio una FragmentTransaction:

```
val frag = MyFragment()  
  
val fragTransaction = supportFragmentManager.beginTransaction()  
  
fragTransaction.add(R.id.fragID, frag)  
  
fragTransaction.commit()
```

# Navigazione in un'app

# Il componente Navigation

- Collezione di librerie e tool, incluso un editor integrato, per creare percorsi di navigazione all'interno di un'app
- Si considera una sola `Activity` per grafo, con molti `Fragment` usati come destinazioni
- Consiste di tre parti principali:
  - Navigation graph
  - Navigation Host (`NavHost`)
  - Navigation Controller (`NavController`)



# Aggiunta delle dipendenze

In `build.gradle (Module)` sotto `dependencies`:

```
implementation
```

```
"androidx.navigation:navigation-fragment-ktx:$nav_version"
```

```
implementation "androidx.navigation:navigation-ui-ktx:$nav_version"
```

# Navigation host (NavHost)

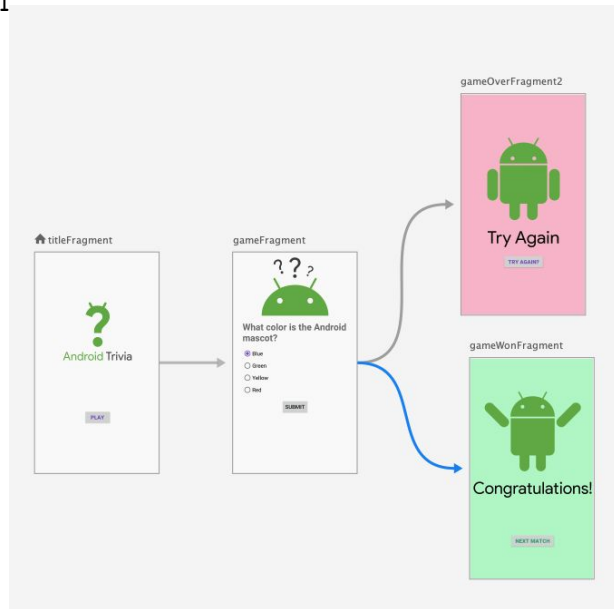
La navigazione avviene caricando di volta in volta un Fragment diverso all'interno di uno speciale contenitore, il NavHost, che gestirà lo swap dei Fragment. Questo va collocato all'interno del layout dell'activity che stia utilizzando.

```
<fragment
    android:id="@+id/nav_host"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:defaultNavHost="true"
    app:navGraph="@navigation/nav_graph_name"/>
```

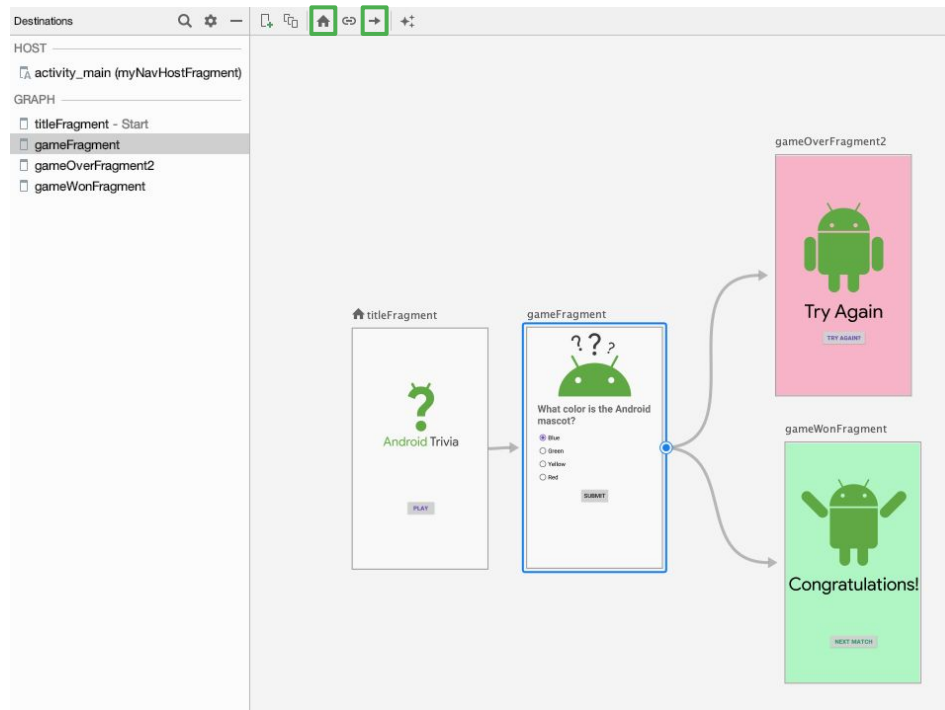
# Navigation graph

Un nuovo tipo di risorsa collocato in `res/navigation`

- E' un file XML contenente i dettagli della navigazione, incluse le destinazioni e le azioni da compiere
- Contiene una lista delle destinazioni (Fragment/Activity) su cui si può navigare e le azioni associate per transitare tra loro
- Opzionalmente contiene l'elenco delle animazioni da visualizzare all'entrata o uscita in/da un fragment



# Navigation Editor in Android Studio



# Creare un Fragment

- Estendiamo la classe `Fragment`
- Sovrascriviamo il metodo `onCreateView()`
- Effettuiamo l'inflating di un layout XML per il Fragment

Nota: si potrebbe fare anche dentro `onCreate` (come per le activity) ma non c'è garanzia che l'Activity sia completamente inizializzata

```
class DetailFragment : Fragment() {  
  
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,  
        savedInstanceState: Bundle?): View? {  
        return inflater.inflate(R.layout.detail_fragment, container, false)  
    }  
}
```

# Specificare il Fragment destinazione

- I Fragment destinazione sono denotati dal tag `action` nel navigation graph.
- Le azioni possono essere definite direttamente in XML o nel Navigation Editor collegando la sorgente alla destinazione
- Gli ID delle azioni vengono autogenerati nella forma `action_<sourceFragment>_to_<destinationFragment>`.

# Esempio di fragment destinazione

```
<fragment
    android:id="@+id/welcomeFragment"
    android:name="com.example.android.navigation.WelcomeFragment"
    android:label="fragment_welcome"
    tools:layout="@layout/fragment_welcome" >

    <action
        android:id="@+id/action_welcomeFragment_to_detailFragment"
        app:destination="@id/detailFragment" />

</fragment>
```

# Navigation Controller (NavController)

- II NavController gestisce la navigazione nel navigation host
  - Specificare un percorso destinazione come action non esegue la navigazione
  - Per seguire un percorso, occorre usare il NavController passandogli un particolare ID di azione, che verrà dunque eseguita



# NavController: esecuzione di un'action

Dentro l'Activity , dopo un evento (ad esempio il click su un bottone), posso eseguire queste istruzioni per chiedere al NavController di eseguire l'action indicata:

```
val navController = this.findNavController(R.id.myNavHostFragment)
navController.navigate(R.id.action_welcomeFragment_to_detailFragment)
```

Invece se la logica è dentro un Fragment, posso navigare in questo modo:

```
view.findNavController().navigate(R.id.action_welcomeFragment_to_detailFragment)
```

# Personalizzare la navigazione

# Passare dati tra destinazioni

Esistono diversi modi per passare parametri tra fragment.

Il modo più semplice è quello di definire variabili all'interno dell'Activity che possano essere lette da ciascun Fragment.

Tuttavia, il meccanismo raccomandato è di usare i **Safe Args** (un plugin Gradle) per passare dati tra una *source* ed un *target* della navigazione.

# Passare dati tra destinazioni

## Safe Args:

- Ci assicurano che gli argomenti abbiano un tipo valido
- Permette di fornire dei valori di default
- Genera una classe `<Source>Directions` con metodi per tutte le azioni in quella destinazione
- Genera una classe per settare gli argomenti per ogni azione
- Genera una classe `<Target>Args` che fornisce accesso agli argomenti della destinazione

# Configurare i Safe Args

Nel file `build.gradle` di progetto:

```
buildscript {  
    repositories {  
        google()  
    }  
    dependencies {  
        classpath "androidx.navigation:navigation-safe-args-gradle-plugin:$nav_version"  
    }  
}
```

Nel file `build.gradle` file dell'app o del modulo:

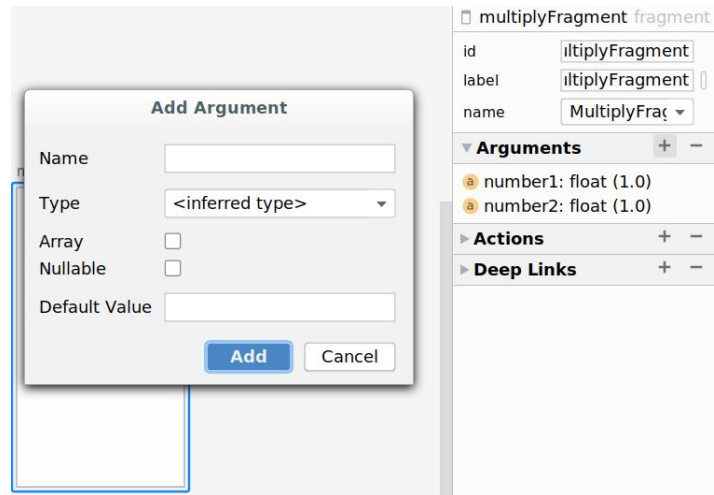
```
plugins{...  
    id 'androidx.navigation.safeargs.kotlin'  
}
```

# Inviare dati ad un Fragment

1. Crea gli argomenti che la destinazione richiede
2. Crea un'azione per collegare la sorgente alla destinazione
3. Setta gli argomenti nei metodi azione su `<Source>Directions`.
4. Naviga secondo quell'azione usando il Navigation Controller.
5. Recupera gli argomenti passati nel Fragment destinazione

# Argomenti del Fragment destinazione

```
<fragment
    android:id="@+id/multiplyFragment"
    android:name="com.example.arithmetic.MultiplyFragment"
    android:label="MultiplyFragment" >
    <argument
        android:name="number1"
        app:argType="float"
        android:defaultValue="1.0" />
    <argument
        android:name="number2"
        app:argType="float"
        android:defaultValue="1.0" />
</fragment>
```



# Tipi di argomenti supportati

Type	Type Syntax <code>app:argType=&lt;type&gt;</code>	Supports Default Values	Supports Null Values
Integer	<code>"integer"</code>	Yes	No
Float	<code>"float"</code>	Yes	No
Long	<code>"long"</code>	Yes	No
Boolean	<code>"boolean"</code>	Yes ( <code>"true"</code> or <code>"false"</code> )	No
String	<code>"string"</code>	Yes	Yes
Array	above type + <code>"[]"</code> (for example, <code>"string[]"</code> <code>"long[]"</code> )	Yes (only <code>"@null"</code> )	Yes
Enum	Fully qualified name of the enum	Yes	No
Resource reference	<code>"reference"</code>	Yes	No



# Tipi di argomenti supportati: classi custom

Type	Type Syntax app:argType=<type>	Supports Default Values	Supports Null Values
Serializable	Fully qualified class name	Yes (only "@null")	Yes
Parcelable	Fully qualified class name	Yes (only "@null")	Yes

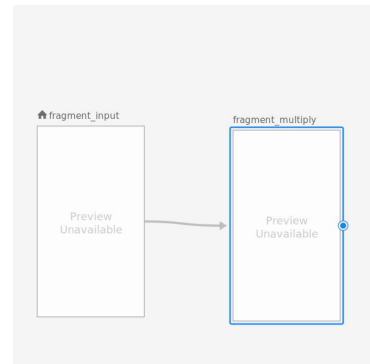
# Creare azioni dalla sorgente alla destinazione

In `nav_graph.xml`:

```
<fragment
    android:id="@+id/fragment_input"
    android:name="com.example.arithmetic.InputFragment">

    <action
        android:id="@+id/action_inputFragment_to_multiplyFragment"
        app:destination="@id/multiplyFragment" />

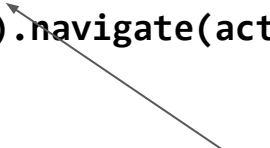
</fragment>
```



# Navigare con le azioni

In `InputFragment.kt`:

```
override fun onCreateView(view: View, savedInstanceState: Bundle?) {  
    super.onCreateView(view, savedInstanceState)  
    binding.button.setOnClickListener {  
        val n1 = binding.number1.text.toString().toFloatOrNull() ?: 0.0  
        val n2 = binding.number2.text.toString().toFloatOrNull() ?: 0.0  
  
        val action = InputFragmentDirections.actionInputFragmentToMultiplyFragment(n1, n2)  
        view.findNavController().navigate(action)  
    }  
}
```



Classe autocostruita da chiamare per passare i parametri

# Recuperare gli argomenti

```
class MultiplyFragment : Fragment() {  
    val args: MultiplyFragmentArgs by navArgs()  
    lateinit var binding: FragmentMultiplyBinding  
    override fun onCreateView(view: View, savedInstanceState: Bundle?) {  
        super.onCreateView(view, savedInstanceState)  
        val number1 = args.number1  
        val number2 = args.number2  
        val result = number1 * number2  
        binding.output.text = "${number1} * ${number2} = ${result}"  
    }  
}
```

Altra classe autocostruita

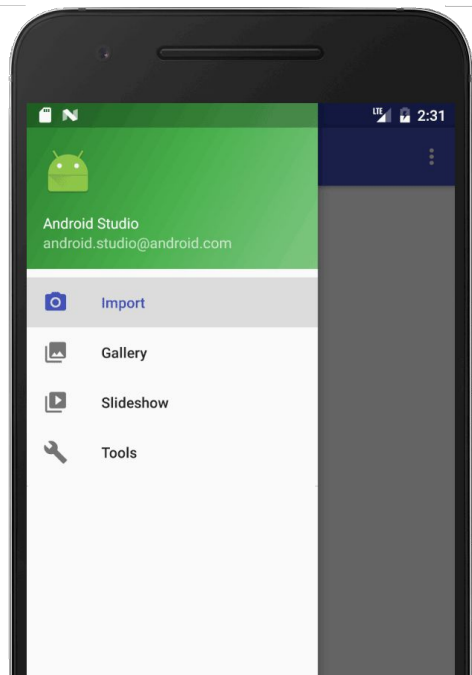
# Interfaccia utente per la navigazione


# Menu rivisitati

Abbiamo già visto come creare delle voci di menu per una Toolbar per navigare tra Activity usando gli Intents. Con questo nuovo componente, è possibile usare la classe `NavigationUI`: se l'ID del menu item è lo stesso di un'azione o di una destinazione, allora `NavigationUI` navigherà verso la destinazione richiesta, semplificando la gestione del menù.

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {  
    val navController = findNavController(R.id.nav_host_fragment)  
    return item.onNavDestinationSelected(navController) ||  
        super.onOptionsItemSelected(item)  
}
```

# Navigation Drawer



Un NavigationDrawer è un pannello che mostra il principale menù di navigazione dell'app. Appare quando l'utente clicca sull'icona del drawer  o quando trascina un dito (swipe) da sinistra destra dello schermo.

Per crearlo occorre:

- Definire un DrawerLayout come elemento root del layout dell'Activity ed inserire un NavigationView
- Collegarlo alla navigazione


# DrawerLayout per il navigation drawer

```
<androidx.drawerlayout.widget.DrawerLayout
    android:id="@+id/drawer_layout" ...>

    <fragment
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:id="@+id/nav_host_fragment" ... />

    <com.google.android.material.navigation.NavigationView
        android:id="@+id/nav_view"
        app:menu="@menu/activity_main_drawer" ... />

</androidx.drawerlayout.widget.DrawerLayout>
```



Qui sono definite le voci di menù



# Ultimi step per il navigation drawer

Connetti il `DrawerLayout` alla navigazione nell'Activity:

```
NavigationUI.setupActionBarWithNavController(this, navController, drawerLayout)
```

Configura il `NavigationView` per usarlo con il `NavController`:

```
NavigationUI.setupWithNavController(binding.navView, navController)
```

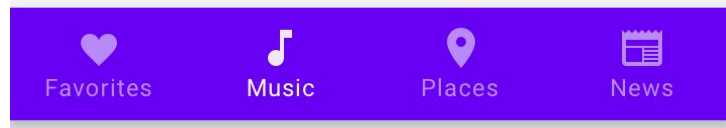
O direttamente

```
binding.navView.setupWithNavController(navController)
```

# Bottom Navigation bar

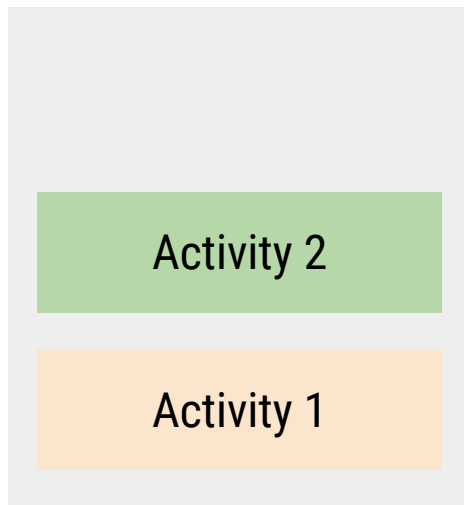
In modo simile è possibile definire una Bottom Navigation bar con bottoni che lanciano Intent o eseguono azioni per navigare verso altri Fragment

<https://developer.android.com/guide/navigation/navigation-ui>



# Capire il back stack

State 1



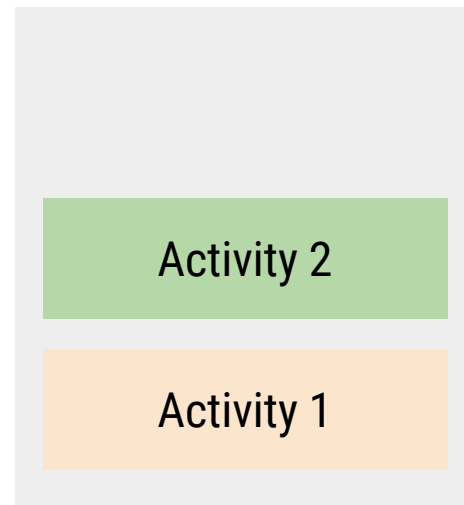
Back stack

State 2



Back stack

State 3



Back stack

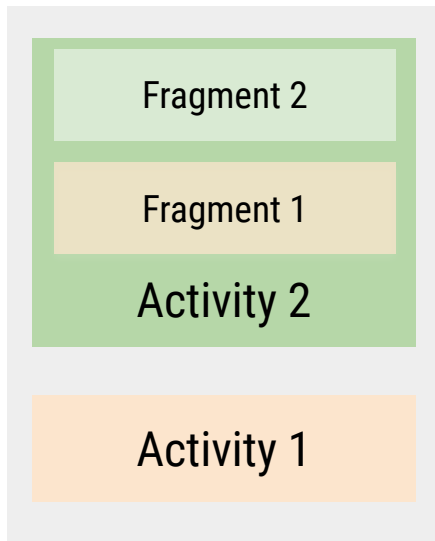
# I fragment ed il back stack

State 1



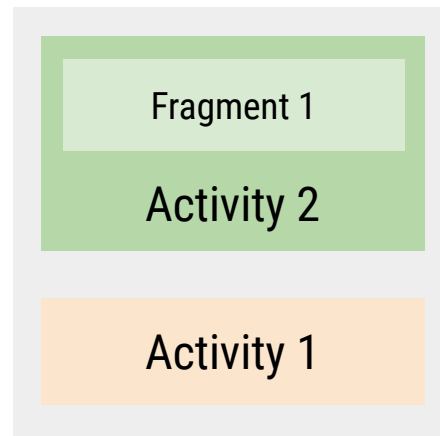
Back stack

State 2



Back stack

State 3

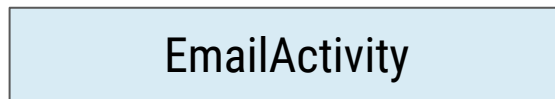


Back stack

# Task e back stack

# Back stack delle Activities

Quando un'app ha diverse Activity, la prima che viene eseguita è l'unica in cima al *back stack*

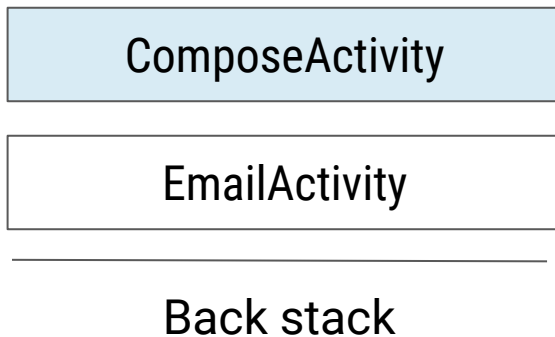


---

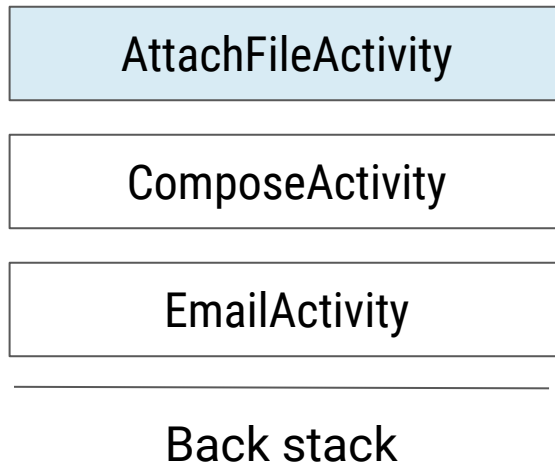
Back stack

# Aggiunta al back stack

Quando viene chiamata un'altra Activity con un Intent, questa viene posizionata sopra la precedente



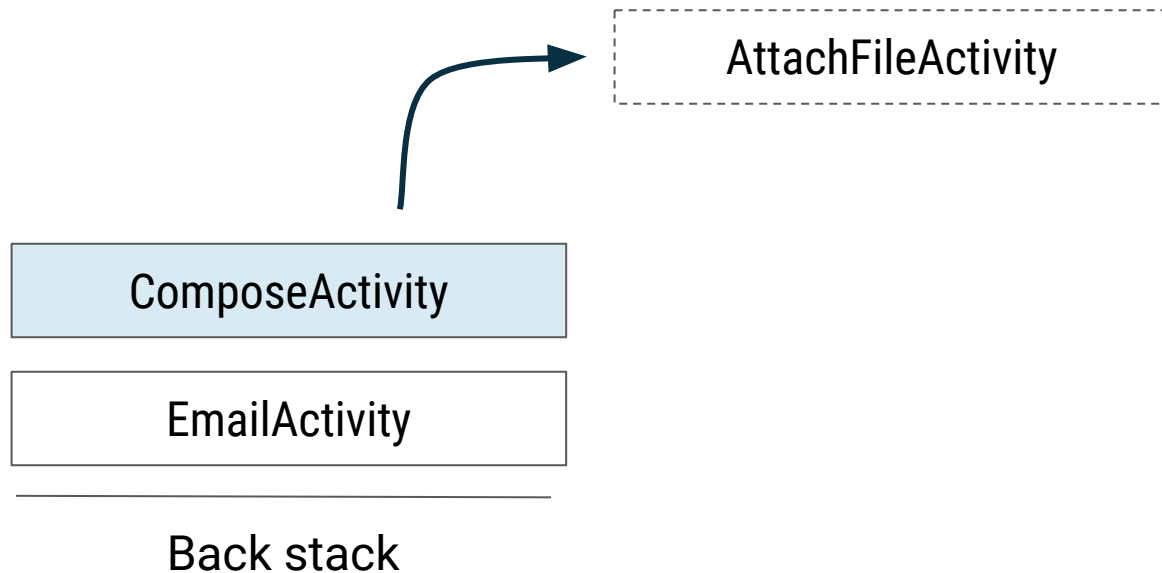
# Aggiunta al back stack



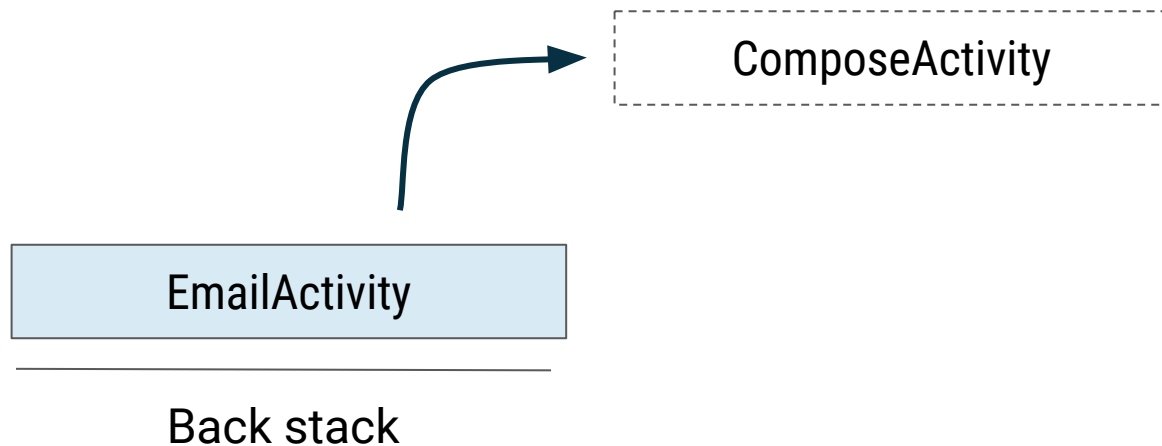


# Cliccare il bottone Back

Se si clicca il bottone "Back" l'ultima Activity è rimossa dal back stack e la penultima torna ad avere il focus



# Cliccare il bottone Back



# Il caso dei Fragment

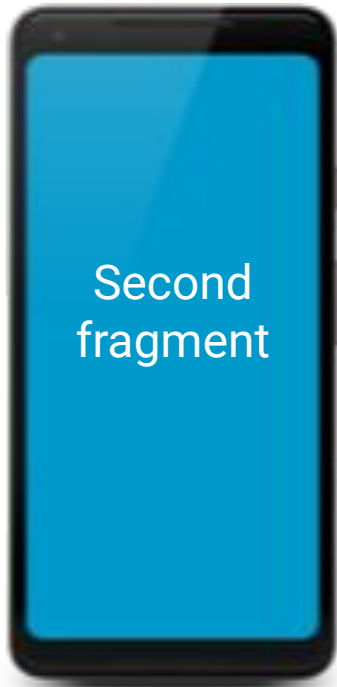


FirstFragment

---

Back stack

# Aggiunta di una destinazione al backstack



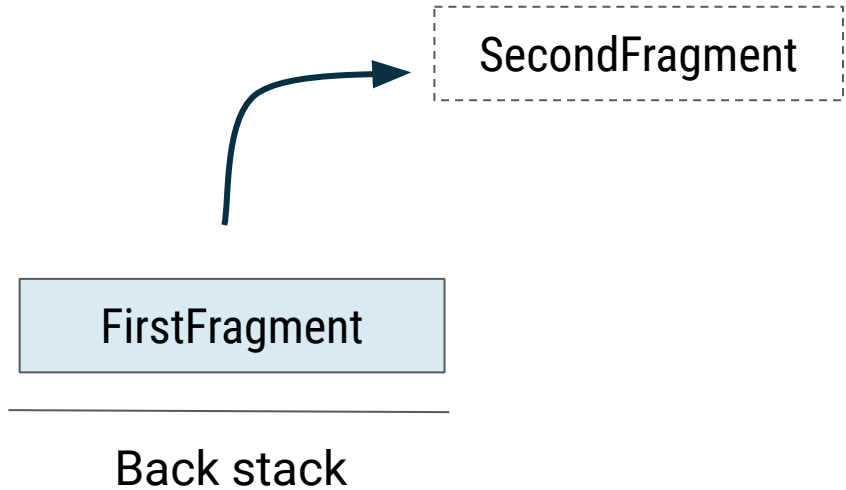
SecondFragment

FirstFragment

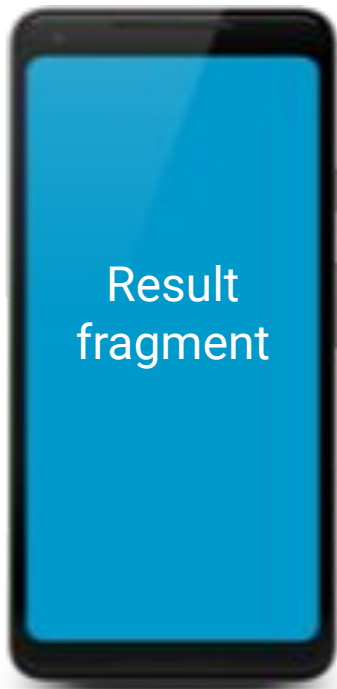
---

Back stack

# Cliccare il bottone Back



# Altro esempio



ResultFragment

Question3Fragment

Question2Fragment

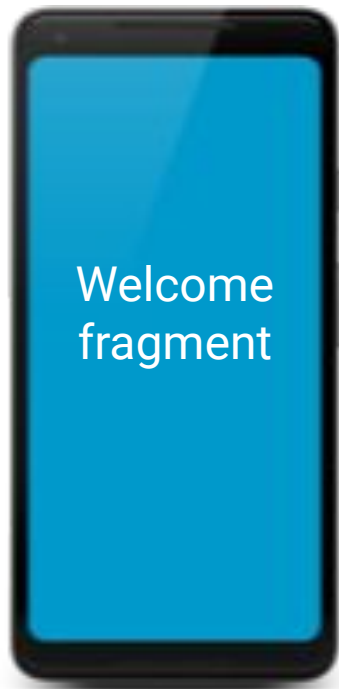
Question1Fragment

WelcomeFragment

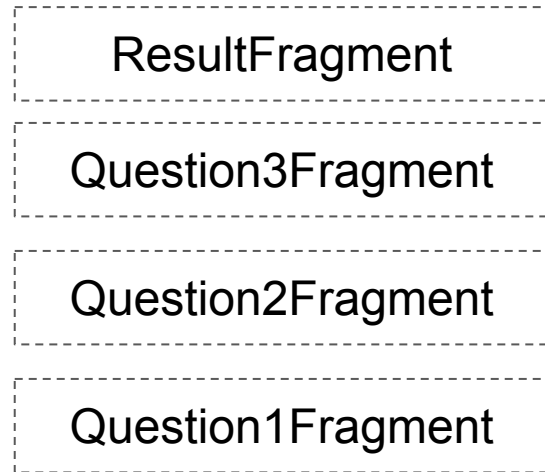
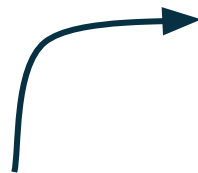
---

Back stack

# Modificare il comportamento di Back



Effettua il “pop” dallo stack di più destinazioni, per tornare ad un punto precedente



WelcomeFragment

Back stack

# Per approfondire

- [Principles of navigation](#)
- [Navigation component](#)
- [Pass data between destinations](#)
- [NavigationUI](#)