

SUPSI

UI Fragments, RecyclerView, ViewPager

Sviluppo di Applicazioni Mobile

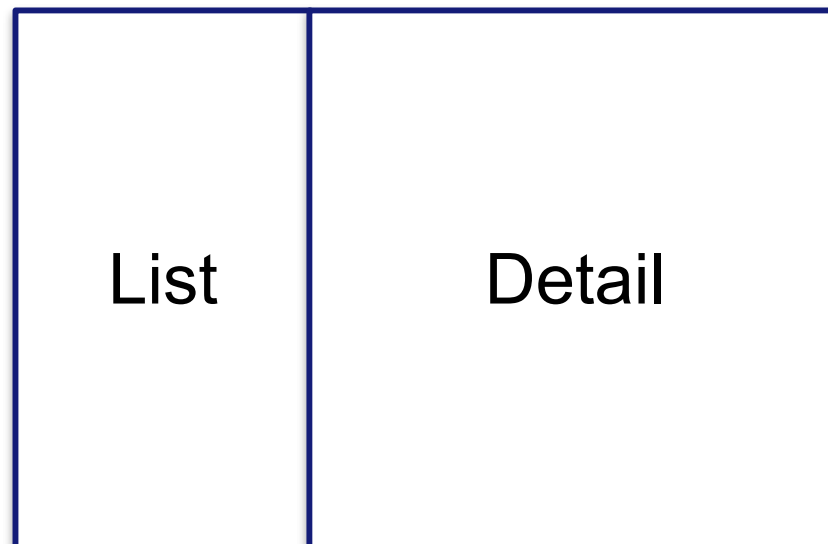
Vanni Galli, lecturer and researcher SUPSI

Obbiettivi

- Imparare ad utilizzare gli *UI Fragments* su Android
- Familiarizzare con il *Fragment Manager*
- Comprendere l'utilizzo pratico del *RecyclerView*

La necessità di avere una UI "flessibile"

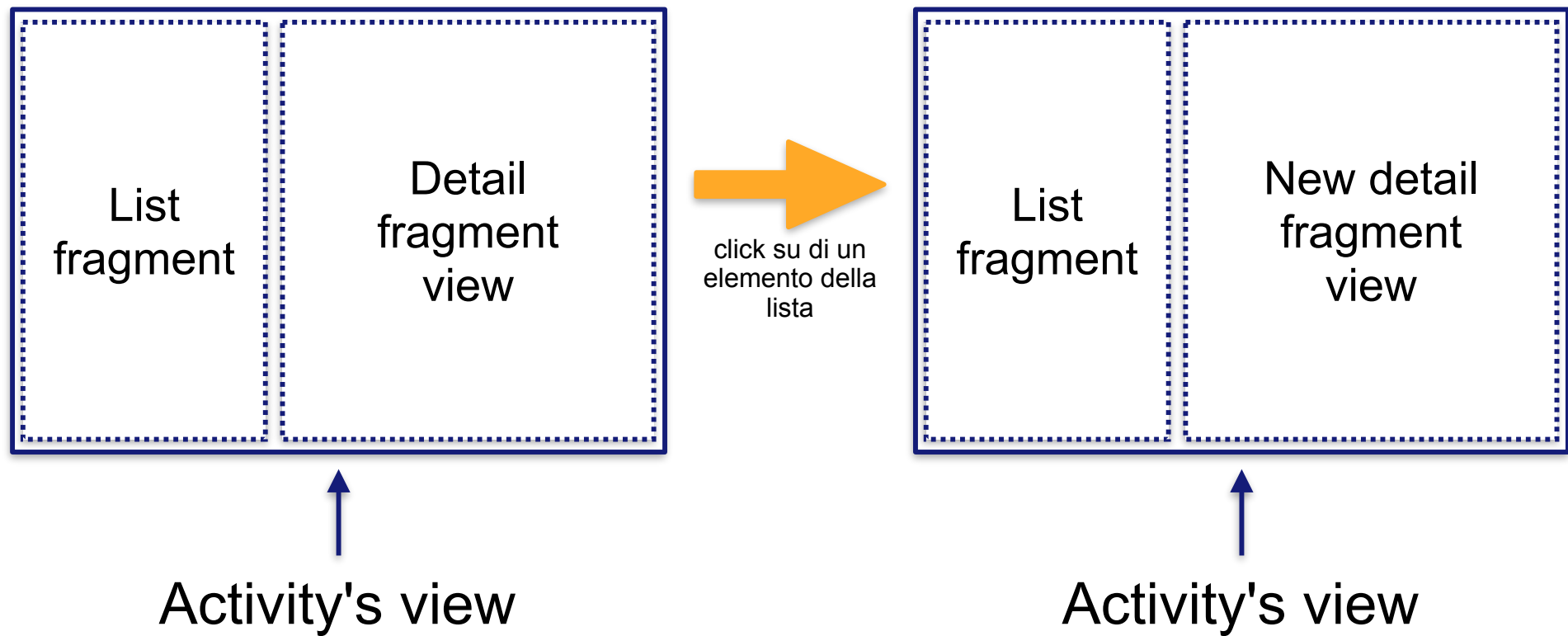
- In varie applicazioni vi è la necessità di avere una UI che si aggiorni man mano, senza che l'activity che la comanda debba per forza essere distrutta e ricreata ogni volta.
- Esempio: applicazione *list-detail* (su tablet); al click di un elemento della lista, va aggiornato il dettaglio:



I fragments

- Un *fragment* è un controller che si sostituisce ad un'activity nell'eseguire determinati task
- Il task principale di un fragment è normalmente la gestione della UI
- Un fragment che gestisce la UI viene chiamato *UI fragment*
- Un UI fragment ha una propria view
- Una view di un'activity contiene uno spazio dove verrà inserito il fragment

Esempio di fragments



Due implementazioni di fragments

- I fragments (introdotti nelle API 11) erano inizialmente disponibili in due implementazioni
 - L'implementazione **nativa** è già compilata all'interno di ogni device; ogni device può dunque contenere una versione leggermente differente dei fragments
 - L'implementazione **support** è una libreria che viene inclusa nella propria applicazione; in questo modo ogni device che esegue l'applicazione usa la stessa implementazione dei fragments
- Queste due implementazioni sono deprecate da Android 9.0 (API 28), e sono state sostituite dalla versione inclusa in *Jetpack (androidx)*
 - A livello di codice non cambia praticamente nulla, cambiano gli import!

Creazione di UI fragment

- I passi per creare uno UI fragment sono gli stessi usati per creare un'activity:
 - comporre la UI definendo dei widgets nel file di layout
 - creare una classe e settare la sua view con il layout definito
 - collegare i widgets del layout al codice

Creazione di UI fragment - Esempio

fragment_new.xml

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

```
<!-- ... -->
```

NewFragment.java

```
public class NewFragment extends Fragment {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        // ...  
    }  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {  
        View v = inflater.inflate(R.layout.fragment_new, container, false);  
        // ...  
        return v;  
    }  
}
```


Il ciclo di vita di un fragment

- Per potere aggiungere un nuovo fragment ad un'activity, questa deve:
 - definire uno spazio, nel proprio layout, dove inserire la view del fragment
 - gestire il ciclo di vita dell'istanza del fragment
- A differenza di un'activity, i metodi per la gestione del ciclo di vita del fragment sono chiamati dall'activity stessa, e non dal sistema operativo (!)
- Il sistema operativo non sa nulla riguardo i fragments che vengono inseriti in un'activity

Due approcci per aggiungere un fragment ad una view

- Ci sono due approcci per aggiungere un fragment ad un'activity:
 - aggiungere il fragment nel **layout** dell'attività
 - aggiungere il fragment nel **codice** dell'attività
- Il primo approccio è semplice ma statico, non permette cioè un controllo a *runtime* del fragment
- Il secondo approccio (che useremo) è quello che permette la vera flessibilità (rimozione di un fragment, sostituzione, ...)

Aggiunta di un fragment ad una view

- Anche se il fragment viene aggiunto nel codice di un'attività, bisogna comunque "fare posto" nella view per poter ospitare il fragment
- Nel caso più semplice, una view di un'attività contiene semplicemente un fragment
- I fragment vengono di norma aggiunti usando una *fluent interface* (vedi prossima slide)

Aggiunta di un fragment ad una view - Esempio

activity_main.xml

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

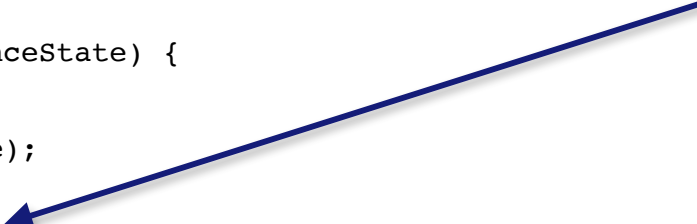
MainActivity.java

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_crime);


    FragmentManager fm = getSupportFragmentManager();
    Fragment fragment = fm.findFragmentById(R.id.fragment_container);

    if (fragment == null) {
        fragment = new MyFragment();
        fm.beginTransaction().add(R.id.fragment_container, fragment).commit();
    }
}
```

richiedo il gestore dei fragments
(androidx, vecchia support library)



le transazioni sono usate per aggiungere,
rimuovere, ... i fragments

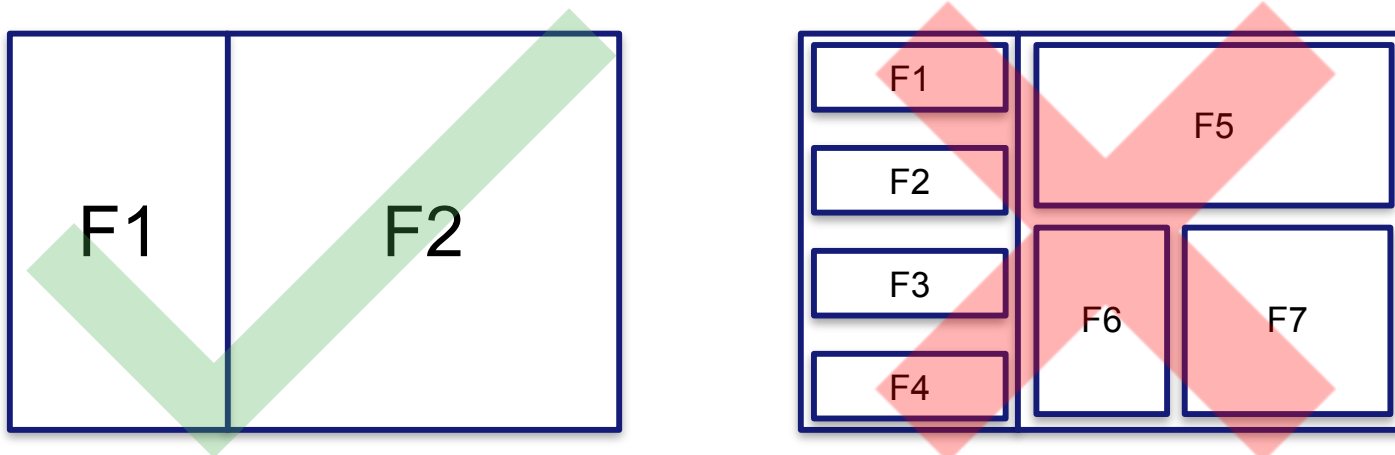


Ancora sull'aggiunta di fragment

- Nel codice precedente viene chiesto al *FragmentManager* di trovare un fragment che un particolare ID, altrimenti di istanziarlo
- Perché un fragment dovrebbe già essere istanziato?
 - Perché la chiamata a `MainActivity.onCreate(Bundle)` potrebbe avvenire in risposta alla nuova creazione dell'attività dopo essere distrutta (ad esempio in caso di rotazione dello schermo)
 - Quando un'activity viene distrutta, il *FragmentManager* salva la lista dei fragments presenti.
 - Quando l'activity viene ricreata il (nuovo) *FragmentManager* recupera la lista di fragment e ristabilisce la situazione che esisteva in precedenza.

Architetture con i fragment

- Utilizzare i fragment correttamente è estremamente importante
- I fragment hanno lo scopo di incapsulare dei "grandi" componenti in modo da favorirne il riutilizzo
- Rule of thumb: un'applicazione non deve mai mostrare più di 2-3 fragment allo stesso tempo



Quando usare i fragments

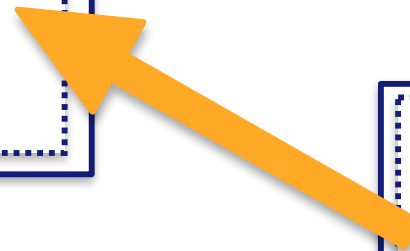
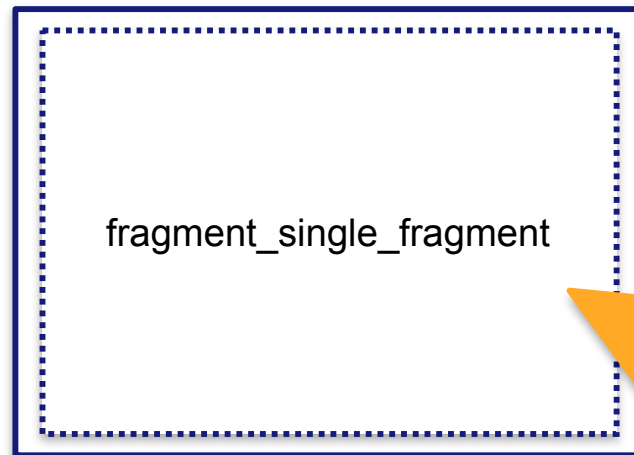
- I fragment sono argomento di discussione nella comunità Android
 - Alcune persone sono contrarie a causa della loro complessità aggiuntiva
- Ci sono però diverse API in Android che si basano sui fragments (ad esempio *ViewPager*), **se si fa uso di queste API si deve dunque fare uso anche dei fragments**
- Anche se non vi è il bisogno di utilizzare API dipendenti dai fragments, i fragments potrebbero diventare utili in applicazioni "grandi"
 - Modificare un'applicazione a posteriori in modo da usare i fragments può però essere molto dispendioso!
- Rule of thumb: **utilizzare sempre i fragments**, a meno che si stia sviluppando un'applicazione piccola, che siamo sicuri non verrà ampliata

Activity con un singolo fragment

- Dal momento che quando si aggiungono fragments ad attività già esistenti si entra in un "campo minato", è possibile sviluppare le proprie applicazioni in modo che ogni singola activity abbia sempre, sin dal principio, almeno un fragment
- È inoltre possibile definire una classe astratta per gestire tutte le attività contenenti un singolo fragment
- Su iCorsi trovate la classe `SingleFragmentActivity.java` e il layout `fragment_single_fragment.xml`
- Un'attività contenente un singolo fragment può estendere *SingleFragmentActivity* e poi fare l'override della funzione *createFragment()*

Activity con un singolo fragment

MainActivity extends SingleFragmentActivity



MyFragment extends Fragment

Fragment con argomenti

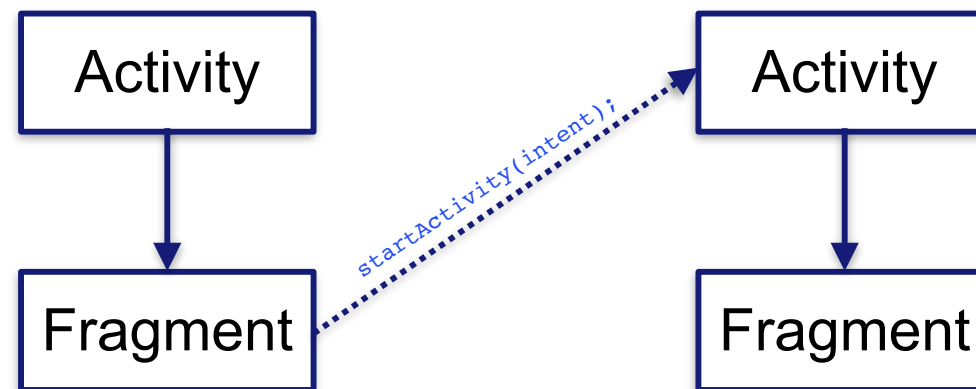
- Il progetto *SingleFragmentActivityExample* mostra come utilizzare correttamente l'astrazione di `SingleFragmentActivity`
- L'attività principale implementa il metodo *createFragment()* per aggiungere il fragment alla propria view
- In molti successivi esempi questa astrazione non più viene utilizzata, in modo da semplificare il codice



Fragment e intent

- Abbiamo visto come un'attività possa lanciarne un'altra
- Un'attività può però anche essere lanciata direttamente da un fragment, il codice risulta pressoché identico:

```
Intent intent = new Intent(getActivity(), MyActivity.class);  
startActivity(intent); // Fragment.startActivity anziché Activity.startActivity
```



Fragment con argomenti

- Ogni istanza di un fragment può contenere un oggetto di tipo *Bundle*
- Il Bundle contiene delle coppie di chiave-valore che funzionano esattamente come gli extras di un'Activity
- Per creare degli argomenti da passare ad un fragment, si crea prima l'oggetto Bundle, e poi si aggiungono i valori voluti:

```
Bundle args = new Bundle();  
args.putSerializable(ARG_MY_OBJECT, myObject);  
args.putInt(ARG_MY_INT, myInt);  
args.putCharSequence(ARG_MY_STRING, myString);
```

Fragment con argomenti

- L'aggiunta di un bundle ad un fragment deve essere fatta dopo la creazione del fragment stesso, ma prima che sia aggiunto ad un'attività
- Per questo motivo i programmatori Android seguono la prassi di creare, all'interno del codice del fragment, un metodo chiamato `newInstance()` che si occupa di istanziare il fragment e gestire il bundle:

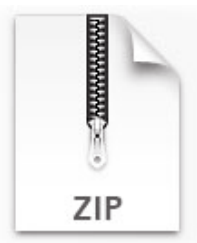
```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // ...
        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment =
            fm.findFragmentById(R.id.fragment_container);

        if (fragment == null) {
            fragment = FragmentOne.newInstance(15);
            fm.beginTransaction().add(R.id.fragment_container,
                fragment).commit();
        }
    }
}
```

```
public class MyFragment extends Fragment {
    // ...
    public static MyFragment newInstance(int esempio) {
        Bundle args = new Bundle();
        args.putSerializable("esempio", esempio);
        MyFragment fragment = new MyFragment();
        fragment.setArguments(args);
        return fragment;
    }
    // ...
}
```

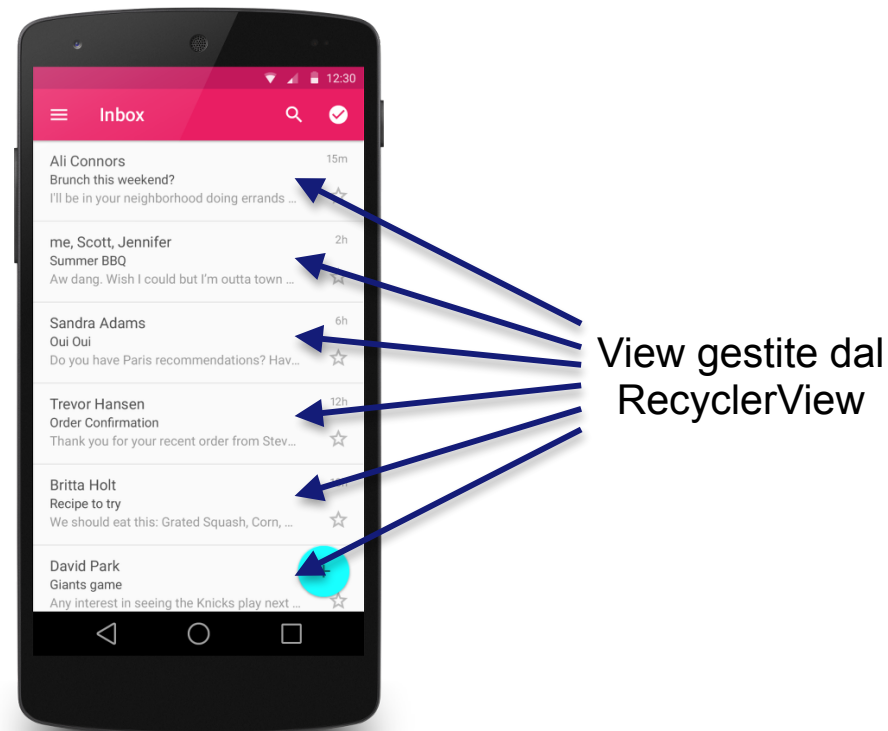
Fragment con argomenti

- Il progetto *FragmentWithArguments* su iCorsi mostra l'utilizzo di fragment con argomenti
- L'attività principale istanza un fragment a cui passa un argomento, che poi viene "riletto" dal fragment stesso



Il RecyclerView

- Nello sviluppo di applicazione mobile capita molto spesso di dover ricorrere a delle liste per mostrare le informazioni desiderate
- Il *RecyclerView* è usato per questo genere di visualizzazione



Visualizzazione di liste con il RecyclerView

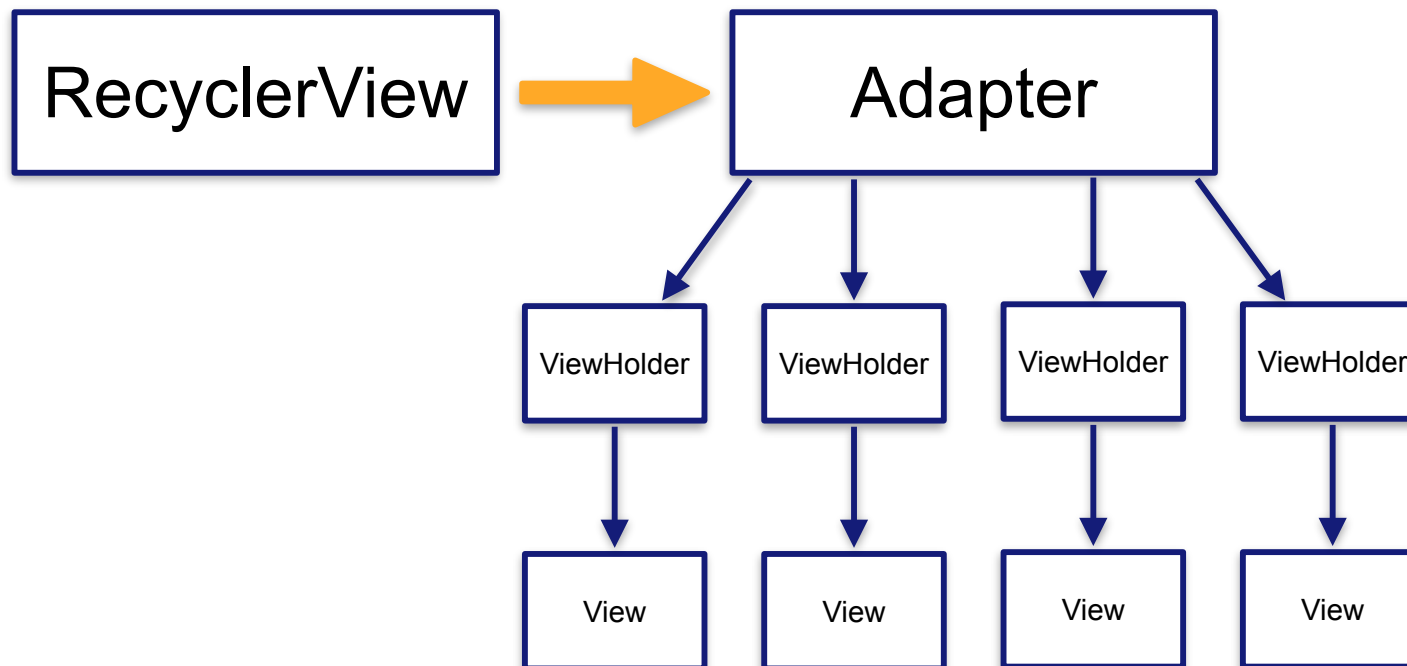
- *RecyclerView* è una sottoclasse di *ViewGroup*
- Il *RecyclerView* in grado di visualizzare una lista di figli di tipo *View*, i quali possono essere molto semplici o complessi
- Il *RecyclerView*:
 - Crea (in memoria) il numero di *View* che appaiono sullo schermo (6 nella slide precedente) e le mostra all'utente
 - Quando un utente scrolla la lista, riusa le *View* che ha in memoria per visualizzare il nuovo contenuto, cioè le **ricicla** continuamente

ViewHolder e Adapter

- L'unica responsabilità del RecyclerView è quella di riciclare le *Views* e di posizionarle sullo schermo
- Per "ottenere" le View, il RecyclerView necessita di due classi che vanno implementate: un *Adapter* e un *ViewHolder*
- Il lavoro del ViewHolder è quello di "tenere" (in inglese "hold") una view
- L'Adapter è invece responsabile di creare i ViewHolders necessari e di fare il *bind* dei ViewHolders ai dati (che sono contenuti nel *Model*)

ViewHolder e Adapter

- Un RecyclerView non crea mai le Views da solo, ma chiede ad un adapter di creare dei ViewHolders, i quali contengono le View



Gli elementi in gioco

MyFragment extends Fragments

```
private RecyclerView mRecyclerView;
private EntryAdapter mAdapter;

@Override
public View onCreateView(...) {
    View view = inflater.inflate(
        R.layout.fragment_list, ...);
    mRecyclerView =
        view.findViewById(R.id.recycler_view);

    mAdapter = new MyAdapter(entries);
    mCRecyclerView.setAdapter(mAdapter);

    return view;
}
```

private class MyAdapter extends RecyclerView.Adapter<MyHolder>

```
private List<Entry> mEntries; // esempio di modello

public MyAdapter(List<Entry> entries) {
    mEntries = entries;
}

@Override
public MyHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    LayoutInflater inflater = LayoutInflater.from(getActivity());
    return new EntryHolder(inflater, parent);
}

@Override
public void onBindViewHolder(EntryHolder holder, int position) {
    Entry entry = mEntries.get(position);
    holder.bind(entry);
}

@Override
public int getItemCount() {
    return mEntries.size();
}
```

fragment_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
...
android:id="@+id/recycler_view"
android:layout_width="match_parent"
android:layout_height="match_parent"/>
```

private class MyHolder extends RecyclerView.ViewHolder

```
private TextView mTitleTextView; // esempio

public MyHolder(LayoutInflater inflater, ViewGroup parent) {
    super(inflater.inflate(R.layout.list_item, parent, false));
    itemView.setOnClickListener(this);

    mTitleTextView = itemView.findViewById(R.id.title);
}

public void bind(Entry entry) {
    mEntry = entry;
    mTitleTextView.setText(mEntry.getTitle());
}
```

list_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
...
<TextView
    android:id="@+id/title"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Title"/>
...
```

Ricaricamento degli elementi

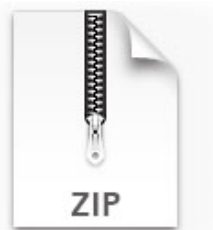
- Quando il modello viene cambiato, l'adapter deve essere informato
- Tipicamente il modello viene cambiato in un'activity diversa da quella corrente (e.g. applicazioni list-details)
- È possibile informare facilmente l'adapter che il modello è cambiato tramite il metodo `notifyDataSetChanged()`:

```
@Override
public void onResume() {
    super.onResume();

    if (mAdapter == null) {
        ...
    }
    else
        mAdapter.notifyDataSetChanged();
}
```

Codice di esempio

- Il progetto *RecyclerViewExample.zip* su iCorsi è un buon punto di partenza per ogni applicazione che utilizzerà un RecyclerView
- Il progetto lancia *MainActivity*, la quale crea il fragment *ListFragment*
- ListFragment implementa un RecyclerView con il suo ViewHolder e il suo Adapter



Il ViewPager

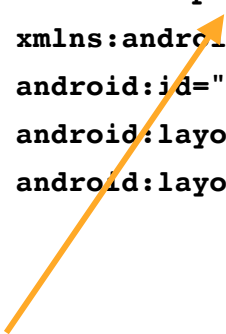
- Il *ViewPager* permette all'utente di navigare attraverso delle view facendo lo "swipe" con il dito (a destra o a sinistra)
- Il ViewPager si comporta in modo "simile" al RecyclerView, richiede cioè anch'esso un *PagerAdapter*
- L'interazione tra ViewPager e PagerAdapter è però più complessa rispetto a quella tra RecyclerView e Adapter
- La classe *FragmentStatePagerAdapter* (sottoclasse di PagerAdapter) si occupa però della maggior parte dei dettagli
- Lo sviluppatore deve prendersi carico di implementare due semplici metodi:
`getCount()` e `getItem(int)`

Implementazione di un ViewPager

- Il ViewPager va dapprima definito (molto semplicemente) all'interno di un layout:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.viewpager.widget.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/my_view_pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

nome del package completo
(nell'implementazione androidx)



Implementazione di un ViewPager

- Nell'activity corrispondente vanno poi implementati 2 metodi:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.crime_pager);
    mViewPager = (ViewPager) findViewById(R.id.my_view_pager);

    ...

    FragmentManager fragmentManager = getSupportFragmentManager();
    mViewPager.setAdapter(new FragmentStatePagerAdapter(fragmentManager) {
        @Override
        public Fragment getItem(int position) {
            MyModel m = modelsHolder.get(position); // esempio!
            return MyFragment.newInstance(m);
        }
        @Override
        public int getCount() {
            return modelsHolder.size(); // esempio!
        }
    });
}
```


Set del primo elemento nel ViewPager (al click sulla lista)

- Oltre all'implementazione dei metodi precedenti, occorre anche informare il *ViewPager* su quale sia il primo elemento da visualizzare
 - Ad esempio: se in un *RecyclerView* clicco sulla terza entry, dovrò visualizzare questa entry nel *ViewPager*
- Questo comportamento può essere facilmente ottenuto nel metodo `onCreate()` dell'activity che si occupa di gestire il *ViewPager*:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...

    // find the current entry in the list of entries and then display it in the ViewPager
    for (int i = 0; i < mEntries.size(); i++) {
        if (mEntries.get(i).getId().equals(entryId)) {
            mViewPager.setCurrentItem(i);
            break;
        }
    }
}
```

Codice di esempio

- Il progetto *RecyclerView+ViewPagerExample.zip* su iCorsi è un'estensione del progetto precedente, in cui è stato aggiunto un ViewPager.
- Al click di un elemento della lista viene lanciata una nuova attività che si occupa di mostrare un fragment contenente un ViewPager.

