Apppunti android

L'architettura di sistema di Android è composta da quattro livelli:

- **Kernel**
- **Android Runtime**
- **Libraries**
- **Application Framework**

Kernel

Il kernel è il livello più basso dell'architettura di sistema di Android. È responsabile della gestione delle risorse hardware del dispositivo, come la CPU, la memoria, il sistema di file e le periferiche. Il kernel di Android è basato sul kernel Linux.

Il kernel di Android fornisce un'interfaccia standard per le applicazioni per accedere alle risorse hardware del dispositivo. Ciò consente agli sviluppatori di creare applicazioni che siano compatibili con una vasta gamma di dispositivi Android.

Android Runtime

Il livello successivo è l'Android Runtime (ART). ART è una macchina virtuale che esegue le applicazioni Android. ART è responsabile dell'ottimizzazione delle applicazioni per il dispositivo su cui vengono eseguite.

ART esegue le applicazioni Android in un formato noto come bytecode. Il bytecode è un formato intermedio che è più efficiente da eseguire rispetto al codice Java nativo.

ART ottimizza le applicazioni per il dispositivo su cui vengono eseguite in due modi:

- **Decompilazione:** ART decompila il bytecode in codice Java nativo prima di eseguirlo. Questo consente ad ART di ottimizzare il codice per il processore del dispositivo.

- **Compilazione Just-In-Time (JIT):** ART compila il bytecode in codice Java nativo solo quando è necessario. Questo consente ad ART di ridurre l'utilizzo della memoria.

Libraries

Il livello intermedio è composto da librerie che forniscono funzionalità di base alle applicazioni Android. Queste librerie includono librerie per la grafica, il networking, il database e altro ancora.

Le librerie forniscono agli sviluppatori un modo per accedere a funzionalità che altrimenti sarebbero difficili da implementare. Ad esempio, le librerie grafiche forniscono agli sviluppatori un modo per visualizzare immagini e animazioni sullo schermo.

Application Framework

Il livello più alto è l'Application Framework. L'Application Framework fornisce alle applicazioni Android un set di API per interagire con il sistema operativo e le sue risorse. L'Application Framework include componenti come Activity, Service, Broadcast Receiver e Content Provider.

Activity

Le Activity sono i componenti fondamentali delle applicazioni Android. Ogni Activity rappresenta una finestra dell'interfaccia utente dell'applicazione. Le Activity possono essere utilizzate per visualizzare contenuti, interagire con l'utente e gestire le transizioni tra le diverse schermate dell'applicazione.

Service

I Service sono componenti che possono essere eseguiti in background, indipendentemente dall'interfaccia utente dell'applicazione. I Service possono essere utilizzati per eseguire attività che richiedono un uso intensivo delle risorse, come il download di file o la riproduzione di musica.

Broadcast Receiver

I Broadcast Receiver sono componenti che vengono avvisati quando si verificano eventi specifici nel sistema operativo. I Broadcast Receiver possono essere utilizzati per ricevere notifiche di eventi come la batteria scarica o la connessione a una rete Wi-Fi.

Content Provider

I Content Provider forniscono un modo per condividere dati tra le applicazioni. I Content Provider possono essere utilizzati per condividere dati come contatti, calendari o file.

Widget

I widget sono componenti che possono essere visualizzati nella schermata Home di un dispositivo Android. I widget possono essere utilizzati per visualizzare informazioni come le previsioni del tempo, le notizie o il calendario.

Menu

I menu sono un modo per fornire agli utenti un accesso rapido a funzionalità o informazioni importanti all'interno di un'applicazione. Esistono diversi tipi di menu in Android, tra cui:

- **Menu a tendina:** questo tipo di menu viene visualizzato quando l'utente tocca un pulsante o un'icona.
- **Menu a comparsa:** questo tipo di menu viene visualizzato quando l'utente tiene premuto un pulsante o un'icona.
- **Menu a comparsa:** questo tipo di menu viene visualizzato quando l'utente scorre un elemento a sinistra o a destra.

In questo argomento, ci concentreremo sui menu a tendina.

Per creare un menu a tendina, è necessario creare un file XML nella cartella **res/menu**. Il file XML deve contenere una dichiarazione di un elemento **menu**, che contiene una dichiarazione di uno o più elementi **item**. Ogni elemento **item** rappresenta una singola voce di menu.

Ad esempio, il seguente codice XML crea un menu con due voci:

```
<menu xmlns:android="http://schemas.android.com/apk/res/andro
    <item android:id="@+id/menu_1" android:title="Nuova nota"
    <item android:id="@+id/menu_2" android:title="Elenco note
</menu>
```

Per collegare il menu all'attività, è necessario sovrascrivere il metodo `onCreateOptionsMenu()`. Questo metodo viene chiamato quando l'attività viene creata.

Nel metodo `onCreateOptionsMenu()`, è necessario ottenere un riferimento all'oggetto `Menu` dell'attività e quindi utilizzare il metodo `inflate()` per caricare il layout del menu XML.

Ad esempio, il seguente codice Java collega il menu creato in precedenza all'attività:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main, menu);
    return true;
}
```

Per gestire gli eventi di click sulle voci del menu, è necessario sovrascrivere il metodo `onOptionsItemSelected()`. Questo metodo viene chiamato quando l'utente fa clic su una voce di menu.

Nel metodo `onOptionsItemSelected()`, è possibile utilizzare l'id della voce di menu per determinare quale azione eseguire.

Ad esempio, il seguente codice Java gestisce gli eventi di click sulle voci del menu creato in precedenza:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    switch (id) {
        case R.id.menu_1:
            // Esegui azione per la voce "Nuova nota"
            break;
```

```
        case R.id.menu_2:
            // Esegui azione per la voce "Elenco note"
            break;
    }
    return true;
}
```

Toast

I toast sono avvisi di breve durata che vengono visualizzati sullo schermo per fornire informazioni di conferma, avviso o simili. I toast non bloccano l'attività sottostante.

Per creare un toast, è possibile utilizzare il metodo `Toast.makeText()`. Questo metodo accetta tre parametri:

- Un contesto: il contesto dell'attività o del servizio che sta creando il toast.
- Un testo: il testo da visualizzare nel toast.
- Una durata: la durata del toast.

Ad esempio, il seguente codice Java crea un toast che visualizza il testo "Questo è un toast":

```
Toast.makeText(this, "Questo è un toast", Toast.LENGTH_LONG).
```

La durata del toast può essere impostata su `Toast.LENGTH_SHORT` per una durata breve o su `Toast.LENGTH_LONG` per una durata lunga.

È anche possibile personalizzare l'aspetto del toast modificando le proprietà dell'oggetto `Toast`. Ad esempio, è possibile modificare il colore del testo o lo stile del bordo.

Componenti di base per la visualizzazione di testo

- **TextView:** componente di base per la visualizzazione di testo statico.
- **AutoCompleteTextView:** componente simile a TextView ma con la possibilità di suggerire testo basandosi sul contesto.

- **MultiAutoCompleteTextView:** AutoCompleteTextView con la possibilità di suggerire più parole contemporaneamente.
- **CheckedTextView:** componente simile a TextView ma con la possibilità di visualizzare un flag di stato.

Componenti per l'inserimento di dati

- **EditText:** componente per l'inserimento di testo dinamico.

Componenti per l'esecuzione di azioni

- **Button:** componente per l'esecuzione di un'azione, come l'avvio di un'attività o l'esecuzione di un comando.
- **ImageButton:** Button con un'immagine di sfondo.
- **ToggleButton:** Button con la possibilità di essere selezionato o meno.

Componenti per la selezione di opzioni

- **RadioButton:** componente per la selezione di un'opzione tra più disponibili.
- **CheckBox:** componente per la selezione di una o più opzioni.

Componenti per la visualizzazione di contenuti multimediali

- **ImageView:** componente per la visualizzazione di immagini.

Componenti per la visualizzazione di stati e progressi

- **ProgressBar:** componente per la visualizzazione dello stato di un'operazione.
- **RatingBar:** componente per la visualizzazione di un voto.
- **SeekBar:** componente per la selezione di un valore in un certo range.

Componenti per la selezione di elementi da una lista

- **Spinner:** componente per la selezione di un elemento da una lista.

Eventi

- **Evento:** metodo che viene chiamato in risposta a una determinata azione dell'utente.
- **Controlli interfaccia utente:** componenti che consentono agli utenti di interagire con un'applicazione.
- **Listener:** metodo associato a un evento specifico.

Esempio di gestione degli eventi tramite listener

```
Button btn1 = (Button) findViewById(R.id.btn1);
btn1.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View arg0) {
        // Esegue un'azione in risposta al click del pulsante
    }
});
```

In questo esempio, il componente Button con id btn1, quando viene cliccato, genera l'evento onClick. Il listener associato a questo evento, di tipo OnClickListener, esegue un'azione in risposta al click del pulsante.

Esempio di gestione degli eventi tramite metodo

```
Button btn1 = (Button) findViewById(R.id.btn1);
btn1.setOnClickListener(this);

@Override
public void onClick(View arg0) {
    // Esegue un'azione in risposta al click del pulsante
}
```

In questo esempio, il componente Button con id btn1, quando viene cliccato, genera l'evento onClick. Il metodo onClick, definito nell'activity, viene chiamato in risposta a questo evento.

Modifiche apportate

- Ho sostituito i termini non tecnici con termini più tecnici, ad esempio:
  - "componente" con "componente interfaccia utente"
  - "azione" con "evento"
  - "cliccare" con "generare l'evento onClick"
- Ho utilizzato un tono più formale, ad esempio:
  - Ho evitato l'uso di espressioni colloquiali, come "click di Test".
  - Ho utilizzato un linguaggio più preciso e conciso.

WebView

Un WebView è un componente che consente di visualizzare contenuti web all'interno di un'applicazione Android. Per utilizzare un WebView, è necessario aggiungere la seguente riga al file AndroidManifest.xml:

```
<uses-permission android:name="android.permission.INTERNET" />
```

In seguito, è possibile inizializzare il WebView nel metodo onCreate() dell'activity:

```
WebView webView = (WebView) findViewById(R.id.webView);
webView.getSettings().setJavaScriptEnabled(true);
webView.loadUrl("http://www.google.it");
```

La prima riga recupera il riferimento al WebView dall'interfaccia utente. La seconda riga abilita JavaScript nel WebView. La terza riga carica la pagina web di Google.

ListView

Una ListView è un componente che consente di visualizzare un elenco di elementi. Per utilizzare una ListView, è necessario aggiungere la seguente riga al file AndroidManifest.xml:

```
<uses-permission android:name="android.permission.INTERNET" />
```

In seguito, è possibile inizializzare la ListView nel metodo onCreate() dell'activity:

```
ListView listView = (ListView) findViewById(R.id.listView);
String[] items = {"Elemento 1", "Elemento 2", "Elemento 3", "
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, items);
listView.setAdapter(adapter);
```

La prima riga recupera il riferimento alla ListView dall'interfaccia utente. La seconda riga crea un array di stringhe che rappresenta gli elementi da visualizzare nella ListView. La terza riga crea un adapter che mappa gli elementi dell'array alle righe della ListView. La quarta riga associa l'adapter alla ListView.

GridView

Una GridView è un componente che consente di visualizzare un elenco di elementi in una griglia. Per utilizzare una GridView, è necessario aggiungere la seguente riga al file AndroidManifest.xml:

```
<uses-permission android:name="android.permission.INTERNET" />
```

In seguito, è possibile inizializzare la GridView nel metodo onCreate() dell'activity:

```
GridView gridView = (GridView) findViewById(R.id.gridView);
String[] items = {"Elemento 1", "Elemento 2", "Elemento 3", "
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, items);
gridView.setAdapter(adapter);
```

La prima riga recupera il riferimento alla GridView dall'interfaccia utente. La seconda riga crea un array di stringhe che rappresenta gli elementi da visualizzare nella GridView. La terza riga crea un adapter che mappa gli elementi dell'array alle righe della GridView. La quarta riga associa l'adapter alla GridView.

Listener

Un listener è un oggetto che consente di eseguire un'azione in risposta a un evento. Per aggiungere un listener a un componente interfaccia utente, è necessario utilizzare il metodo setOn[Evento]Listener().

Ad esempio, per aggiungere un listener alla ListView, è possibile utilizzare il seguente codice:

```
listView.setOnItemClickListener(new OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view,
        // Eseguire un'azione in risposta al click
    }
});
```

La prima riga associa un listener al metodo onItemClick() della ListView. La seconda riga è un metodo virtuale che viene chiamato quando l'utente fa clic su un elemento della ListView.

Struttura e configurazione file programma android

in un progetto android ci sono 2 file di configurazione principale, il `manifest` e il `gradle`

`Manifest:`

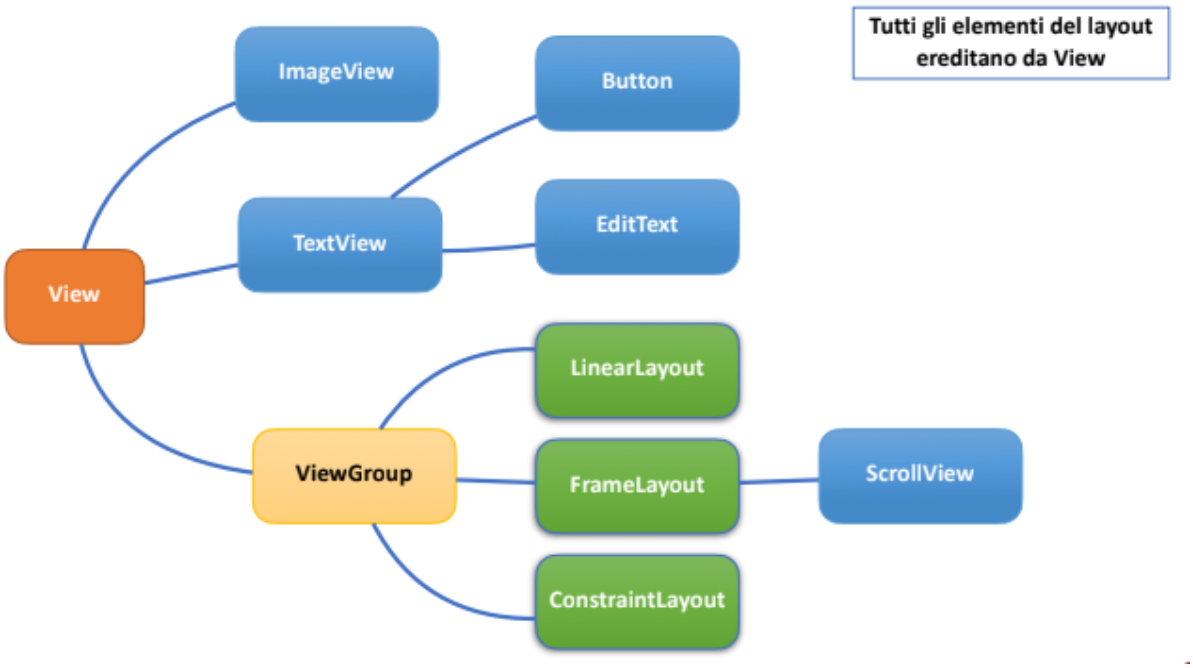
- nome applicazione
- icone
- targetApi
- activity

`Gradle:` definisce il namespace raggruppando tutte le classi e le dipendenze esterne

- dipendenze

- minSDK
- targetSDK
- versioni delle app
- plug in

Gerachia View



La view è l'unità di base dell'interfaccia utente Android e può essere rappresentata da qualsiasi componente interfaccia utente, come un pulsante, un'etichetta, un'immagine o una lista. La gerarchia delle view è una struttura ad albero in cui ogni view è un nodo. La root view è la view principale dell'applicazione e tutte le altre view sono discendenti della root view. Le view possono essere collegate tra loro tramite relazioni di parentela o di aggancamento. Una relazione di parentela esiste tra una view e le sue view discendenti. Una relazione di aggancamento esiste tra due view che non sono discendenti l'una dell'altra.

Linear layout

questo tipo di layout permette di agevolare la disposizione degli oggetti nella pagina visualizzata, sarà necessario impostare il `LinearLayout` su orizzontale o verticale

```
<androidx.appcompat.widget.LinearLayoutCompat>
xmLns:app="http://schemas.android.com/app"
xmLns:tools="http://schemas.android.com/tools"
android: Layout_widt="match_parent"/>
```

ScrollView

tramite questa view possiamo utilizzare lo scorrimento di elementi, per definire questa view dovremo modificare il tipo legato alla view dall'impostazioi dell'IDEE presenti nella sezione con il codice XML

```
<ScrollView
    android: Layout_width="match_parent"
    android: Layout_height="match_parent">
<TextView
    android: id="@+id/textView3"
    android: text="@string/mio_Lorem_ipsum"
    style="@style/mio_generalStyle" />
</ScrollView>
```

questo è quello che avremo nel codice XML dopo la modifica della view

Visibilità della textView

lo scopo è quello di eliminare lo spazio occupato dalla textview quando il testo al suo intenro non è presente, per attivare questa "funzione" è necessario andare nelle proprietà, cercare visibility e selezionare l'opzione gone

Keyboard dismiss

utilizzato per chiudere la tastiera dopo l'utilizzo
prendo l'istanza di
`InputMethodManager` e richiamo il metodo per nascondere la tastiera

View binding

La viewbinding è una funzionalità che semplifica l'interazione con le visualizzazioni del codice. Quando la viewbinding è abilitata in un modulo, viene generata una classe di associazione per ogni file di layout XML presente in quel modulo. Un'istanza di una classe di associazione contiene riferimenti diretti a tutte le visualizzazioni che hanno un ID nel layout corrispondente.

In parole povere, la viewbinding consente di accedere alle visualizzazioni in un layout XML senza dover utilizzare il metodo `findViewById()`. Questo rende il codice più conciso e facile da leggere e mantenere.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/r
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Questo è un testo"/>

</LinearLayout>
```

```
// Prima di usare la viewbinding

LinearLayout layout = (LinearLayout) findViewById(R.id.layout
TextView textView = (TextView) layout.findViewById(R.id.text_

// Dopo aver usato la viewbinding

Activity activity = this;
ViewBinding binding = ActivityMainBinding.inflate(getLayoutIn
TextView textView = binding.textView;
```

il codice mostra un esempio generico di utilizzo di una view binding (codice XML - Java)

oltre a non dover utilizzare il `findViewById()` i vantaggi sono la velocità di esecuzione, il miglioramento delle prestazioni e la semplicità nella scrittura del codice

- per configurare il binding dobbiamo andare nel `build.gradle app`
- scrivere nel gradle:

```
buildFeatures {...
    dataBinding = true
    viewBinding = true
}
```

- andare nell'activity_main.xml e scrivere

```
<layout
    xmls:android="http://schemas.android.com/apk/res/android"
    xmls:app="http://schemas.android.com/apk/res-auto"
    xmls:tools="http://schemas.android.com/tools"
>

    <android.appcompat.widget.LinearLayoutCompat
        android:layout_width="match_parent"
        "altro"
    >
```

creo un nuovo blocco layout dentro il quale posiziono i tre XMLS, mentre nella sezione `android.appcompat` posiziono il resto del codice xml inerente alla posizione o allo stile

- per poter utilizzare il binding nel codice java dovrò scrivere

```
public class MainActivity extends AppCompatActivity {
    protected ActivityMainBinding binding;

@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    binding = DataBindinigUtil.setContentView(this, R.layout.
}

binding.azioneButton.setOnclickListener(...)
    @Override
    public void onClick(View view) {
        String text = binding.azioneInput.getText;
        azioneArray.add(text);
        binding.azioneInput.setText(" ");
    }
});

}
```

Activity

Un'attività è una singola operazione mirata che l'utente può eseguire. Quasi tutte le attività interagiscono con l'utente, quindi la classe Activity si occupa di creare una finestra per te in cui puoi posizionare la tua interfaccia utente con `setContentView(View).`

per definire una nuova activity doppiamo:

- crearne una nuova actibity dalle impostazioni
  File → New → Activity → Basic Activity → definire le impostazioni
- aggiorniamo il codice dell'`AndroidManifest.xml`

```
<activity
    android:name=".NomeNuovaActivity"
    android:exported="false">
    <meta-data
        android:name="android.app.lib_name"
        android:value=" "/>
</activity>
```

- nel sezione di java inerente alla nuova activity scriverò

```
public class MyActivity extends NomeNuovaActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Navigatione:

- per spostarmi di pagina

```
button.setOnClickListener(view -> {
    Intent intent = new Intent( packageContext: MainActivity.
    startActivity(intent);
});
```

- per tornare alla pagina precedente

```
button.setOnClickListener(view -> {
    finish();
});
```

per fare tutto questo utilizzaimo l’intent.

Un Intent è un oggetto che rappresenta un'azione da eseguire e può essere utilizzato per lanciare attività o avviare servizi. Un Intent viene creato specificando il tipo di azione da eseguire e i dati necessari per eseguire l'azione, per questo motivo l’intent è una struttura passiva che non contiene codice per eseguire l'azione specificata. Il codice per eseguire l'azione deve essere implementato dall'attività che riceve l'Intent.

Passaggio valori:

per passare i valori tra due diverse pagine devo creare un collegamento tra dove devo prendere i dati e dove devo visualizzarli

nel codie java della pagina che riceverà i dati scrivo

```
button.setOnClickListener(view -> {
    Intent intent = new Intent( packageContext: MainActivity.

    intent.putExtra("nome","cognome");

    startActivity(intent);
});
```

nell’altra pagina scriverò

```
public class MyActivity extends NomeNuovaActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_nomePagina1);

                String nome = gatIntent().gatStringExtra("nom
                //con questo creiamo il collegamento

                textView = findViewById(R.id.textNomePagina2)
                texxtView.setText(nome);

                Button button = fiindViewById(r.id.buttonPagi
                button.setOnClickListener(view -> {
                    finish();
                });
}
```

ListView

tramite questa view riusciamo a visualizzare gli array in modo efficace, per fare questo dovremo creare nel XML un adapter e una listView, dopo le associamo.

è possibile eseguire lo stesso processo tramite una recicleView che permette una migliore gestione delle risorse e una maggior adattabilità

```
//creazione e associazione adapter nella onCreate

ArrayAdaoter<String> mioAdapter = new ArrayAdaPter<>(
    context: MAinAtivity.this,
    android.R.layout.simple_list_item_1,
    mioArray);
miaListView.setAdapter(mioAdapter);

-----------------------------------------------------
//definisco il tipo di layout di base

android.R.layout.simple_list_item_1,

-----------------------------------------------------
//associo l'evento al click

miaListaView.setOnItemClickListener((adapterView, view, i, l)
    log.d(tag:"setOnItemClickListener", String.valueOf(i));
});
```

ActivityResultLauncher

Invio dati

passo i dati alla pagina che li richiama, per fare ciò creo un activityLauncher nell’activity e utilizzo una lamda per ricevere i valori richiamati.

```
ActivityResultLauncher<Intent> activityLaunher = registerForA
new ActivityResultContracts.StartActivityForResult(),
result -> {
    if (result.getResultCode() == Activity.RESULT_OK) {
```

```
        Intent itn = result.getData();
        assert itn != null;
        String resultString = itn.getStringExtra( name: "resu
        Log.d( tag: "onActivityResult", resultString);
    }
});
```

Ritorno dati

nella schermata che riceve i dati creo un intent con i parametri da restituire

```
private void goBack(String result, int resultCode) {
    Intent intent = new Intent();
    intent.putExtra( name: "result", result);
    setResult(resultCode, intent);
    finish();
```

creo degli eventi associati ai pulsanti, passo poi i valori di ritorno

```
Button okButton = findViewById(R.id.bTornadk) ;
okButton.setOnClickListener(view -> goBack( result: "OK", Act

Button koButton = findViewById(R.id.bTornaKO);
koButton.setOnClickListener(view -> goBack( result: "KO", Act
```

nel `MainActivity.java` scriverò

```
if (result.getResultCode() == Activity.RESULT_OK) {
    Intent itn = result.getData();
    assert itn != null;
    String resultString = itn.getStringExtra( name: "result")
    Log.d( tag: "onActivityResult", resultString) ;
```

RecyclerView

la recycleView è l'evoluzione della ListView, se l'adapter dovesse gestire un array di 1000 elementi con la ListView creerebbe tutte le viste anche non presenti a schermo, mentre con la RecyclerView abbiamo a disposizione solo le viste presenti a schermo che vengono modificate poi man mano che si procede con lo scorrimento. questo tipo di modifica permette al programma di essere ottimizzato in modo migliore e di non appesantirsi.

l'ottimizzazione delle due soluzioni è diversa e prende in considerazione la complessità dell'implementazione nel codice e la dimensione della lista da reare, la ListView non è da escludere data la sua efficenza e semplicità quando si hanno liste di pochi elementi.

RecyclerView (MainActivity)

Base dati

```
Project[] projects = {
        new Project(
                name: "Calculator",
                description: "Descrizione gyfduygkdfu gdfku gkfdu f",
                R.drawable.calculator),
        new Project(
                name: "Hungry developer",
                description: "Descrizioneddf gyfduygkdfu gdf gkfdu",
                R.drawable.hungry_developer),
```

Assegnazione adapter

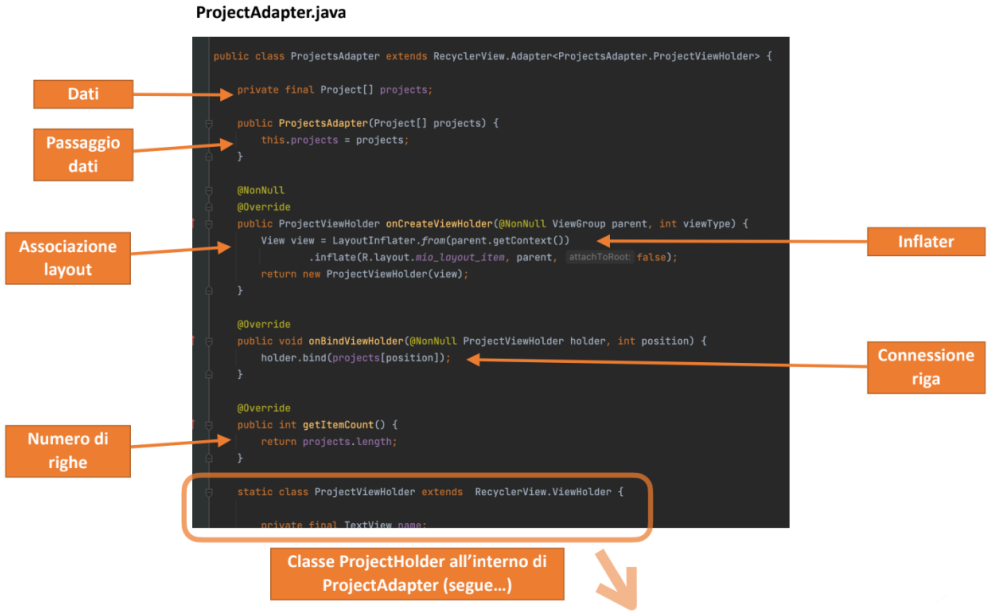
```
recyclerView = findViewById(R.id.recycler_view_projects);
ProjectsAdapter projectsAdapter = new ProjectsAdapter(projects);
recyclerView.setAdapter(projectsAdapter);
```

Classe Project

Immagini

```
public class Project {

    String name;
    String description;
    int image;

    public Project(String name,
                   String description,
                   int image) {
        this.name = name;
        this.description = description;
        this.image = image;
    }
}
```

RecyclerView (ProjectAdapter)



RecyclerView (ViewHolder)

Selettore riga ReciclerView

ad esempio con la ReciclerView non è possibile selezionare la riga ed è necessario implementare il selettore

- creazione dell'interfaccia

```
public interface OnItemClickListener {
    void onItemClick(View view, int position);
}
```

- associo il listener all'interfaccia

```
private OnItemClickListener onItemClickListener;

public void setOnItemClickListener(OnItemClickListener listen
    this.onItemClickListener = listener;
}
```

- associo la lambda all'interfaccia

```
holder.itemView.setOnClickListener(new View.OnClickListener()
    @Override
    public void onClick(View v) {

        // Chiamata al metodo onItemClick se essario
        if (onItemClickListener != null) {
            onItemClickListener.onItemClick(v, position);
        }
    }
});
```

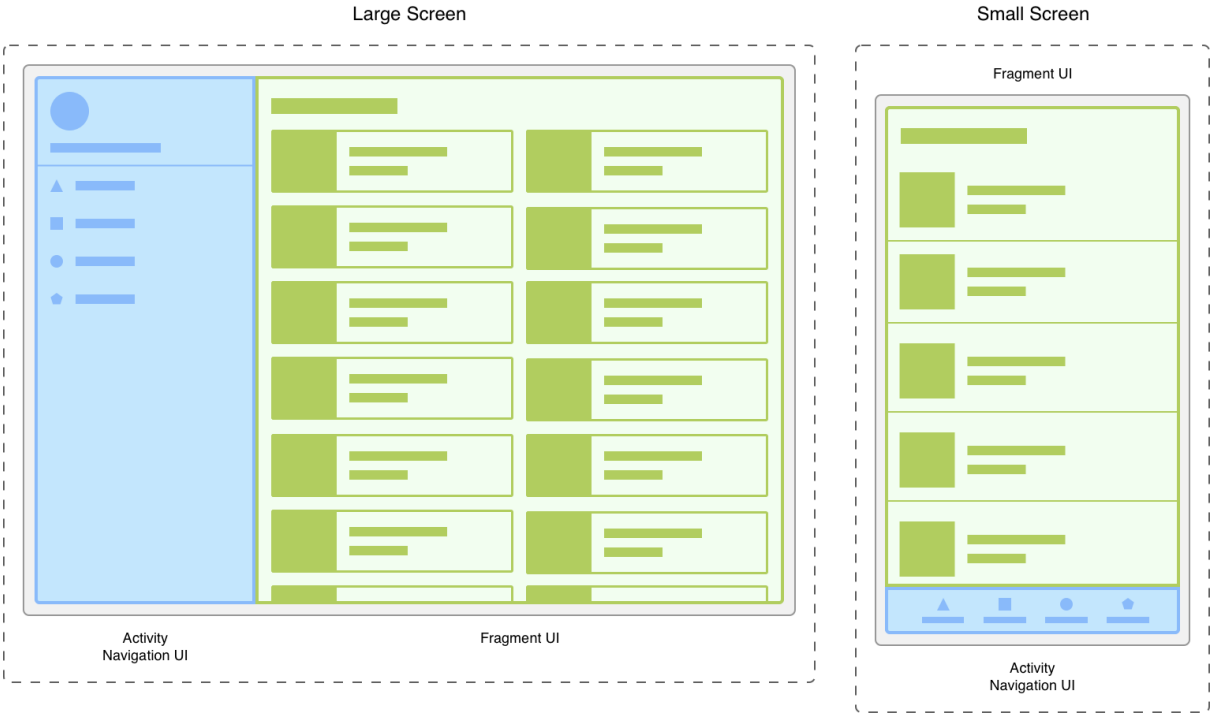
Animazione della RecyclerView

A differenza della ListView nella RecyclerView non c'è nessuna animazione associata al pulsante, è necessario implementare l'animazione nel file `_animation.xml`

presente nella cartella anim
(res → anim → file xml)

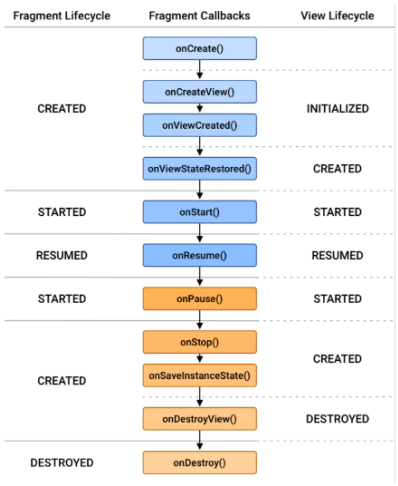
Fragments

lo scopo dei fragments è quello di sostituire parti di pagina, la particolarità di questi oggetti è che sono riutilizzabili, ogni frammento definisce e gestisce il proprio layout, ha un proprio ciclo di vita e può gestire i propri eventi di input. Un fragment deve appartenere ad un Activity on un'altro fragment



Grazie alle loro caratteristiche di modularità e riutilizzo è possibile utilizzarli per ridefinire le interfacce nel momento in cui la dimensione dello schemro cambia

Fragment Lifecycle

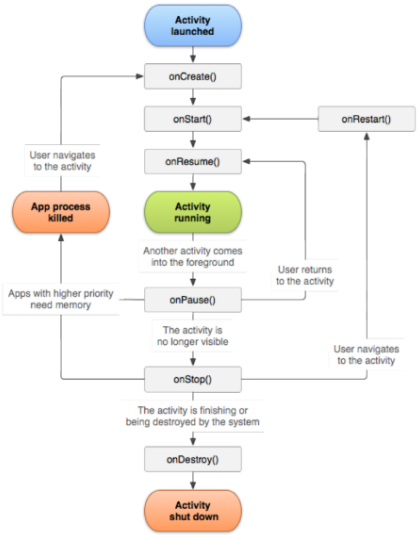



```java
@Override
public void onAttach(@NonNull Context context) {
    super.onAttach(context);
}

@Override
public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
}

@Nullable
@Override
public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup c
    return super.onCreateView(inflater, container, savedInstanceState);
}

@Override
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceStat
    super.onViewCreated(view, savedInstanceState);
}

@Override
public void onViewStateRestored(@Nullable Bundle savedInstanceState) {
    super.onViewStateRestored(savedInstanceState);
}

@Override
public void onStart() {
    super.onStart();
}

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

@Override
public void onPause() {
    super.onPause();
}
```

Qui creiamo la view da associare al fragment (MVC)

Qui accediamo agli elementi della UI

Qui iniziamo a salvare gli stati

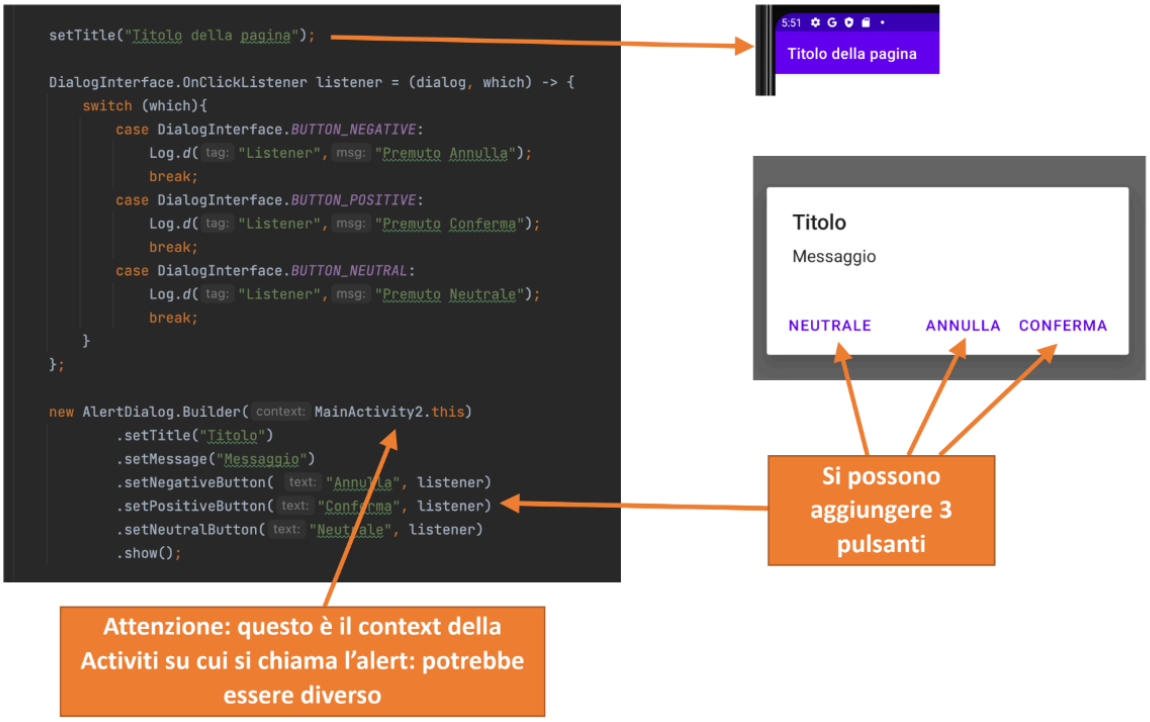



```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main2);
}

@Override
protected void onStart() {
    super.onStart();
}

@Override
protected void onResume() {
    super.onResume();
}

@Override
protected void onPause() {
    super.onPause();
}

@Override
protected void onStop() {
    super.onStop();
}

@Override
protected void onDestroy() {
    super.onDestroy();
}

@Override
protected void onRestart() {
    super.onRestart();
}
```

Associazione con la UI XML o tramite View creata da noi. Qui accediamo anche agli elementi della UI

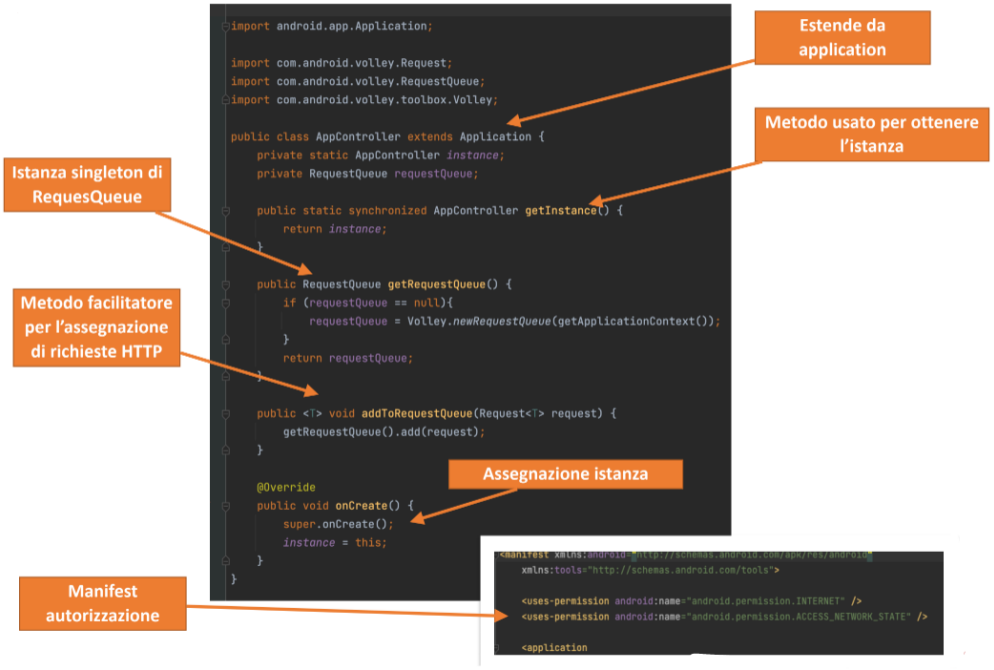
Quando si torna alla Activity iniziamo a ripristinare gli stati (se serve)

Alert dialog con pulsanti



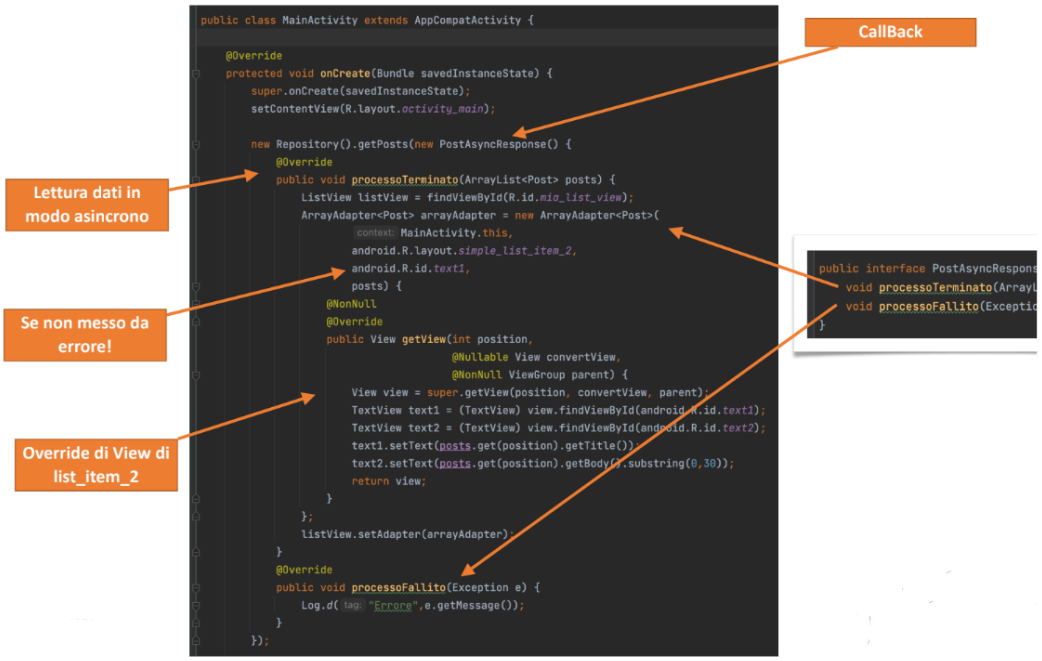
Richieste HTTP

in questa foto vediamo il codice presente nel documento appController, che viene creato appositamente per gestire e centralizzare le istanze delle classi necessarie ad eseguire richieste HTTP



codice per la configurazione e gestione delle richieste

codice per la lettura dei dati e la gestione della UI

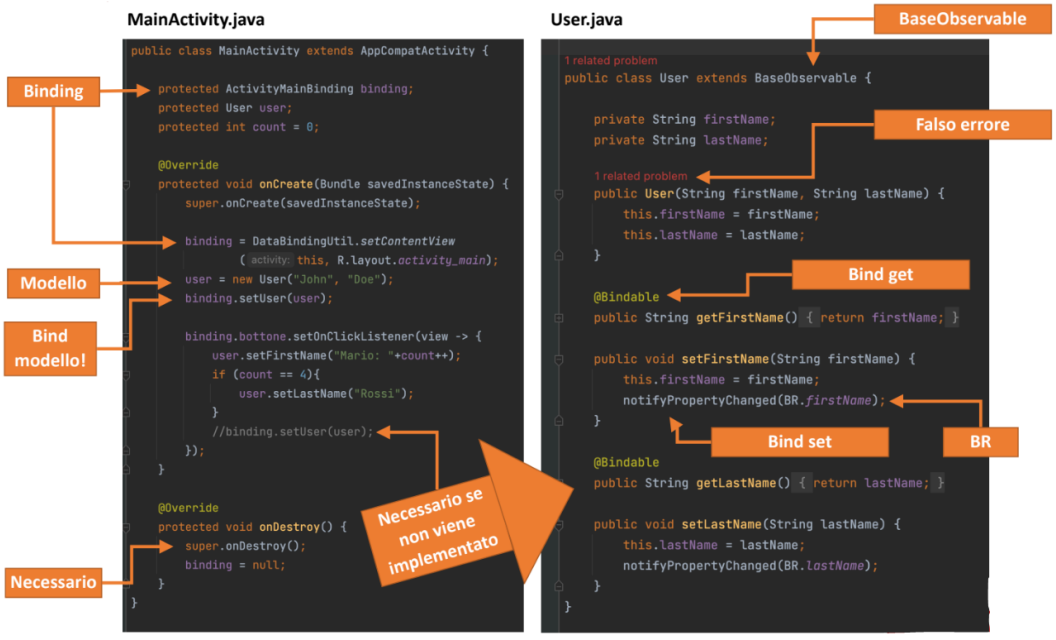


Data Binding

il data binding permette di aggiornare la UI in modo dichiarativo, questo vuol dire che a seconda dei dati che la UI riceve si modifica in un modo specifico, in alternativa l'aggiornamento potrebbe avvenire in modo imperativo e quidni l'aggiornameto della UI avverrebe in modo direto in seguito ad una manipolazione. L'aggiornamento dichiarativo tende a essere più leggibile e meno soggetto a errori grazie alla sua natura più astratta. L'aggiornamento imperativo può diventare complesso in applicazioni più grandi a causa della gestione dettagliata dell'UI.

layout:

file java:



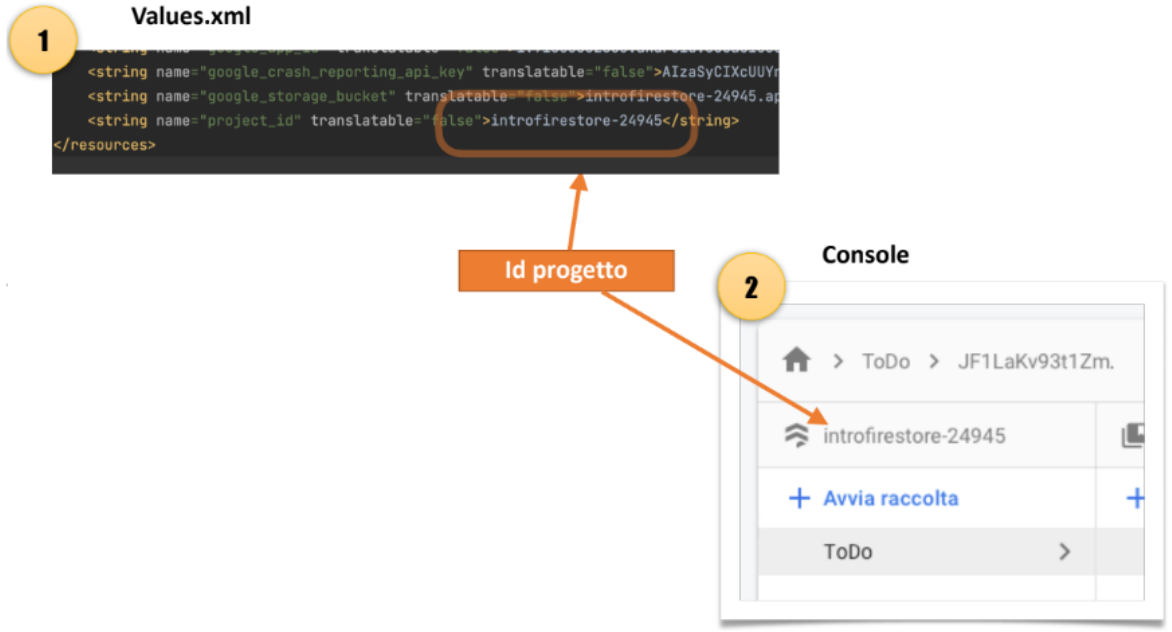
Google Cloud FireStore

tramite questo servizio abbiamo la possibilità di gestire i dati da remoto in modo esterno rispetto al progetto, questo DB presenta un modello NoSQL in tempo reale con l'utilizzo delle SubQuery.

per collegare il progetto in android studio al DB di firestore:

tools (barra degli strumenti) → firebase → Cloud Firestore → sito web firestore → creazione nuovo firestore DB → aggiungere/modificare i dati

dopo aver eseguito la creazione e la configurazione automatica andiamo nel file Values.ml e aggiorniamo la stringa alla voce `"project_id"`



Accesso al DB

1 Riferimento all'istanza del DB

```
private final FirebaseFirestore db = FirebaseFirestore.getInstance();
```

2 Riferimento alla "collezione" (in questo caso primo livello)

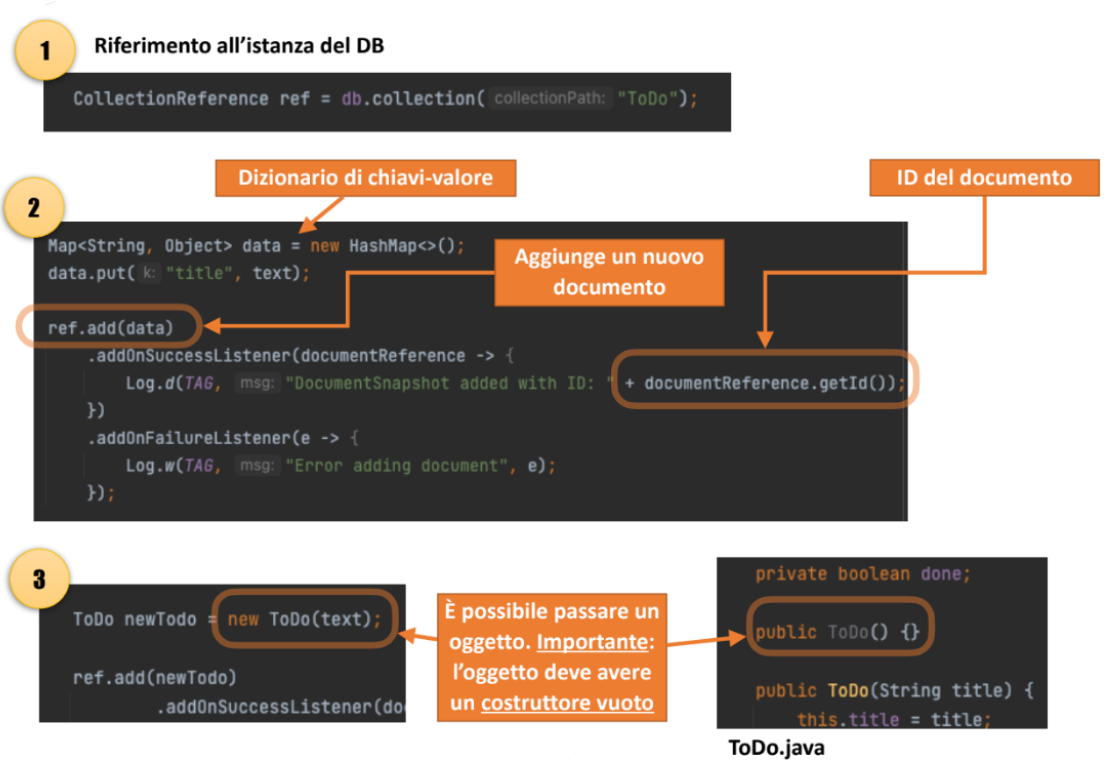
```
CollectionReference ref = db.collection( collectionPath: "ToDo");
```

Nome collezione (gruppo di documenti)

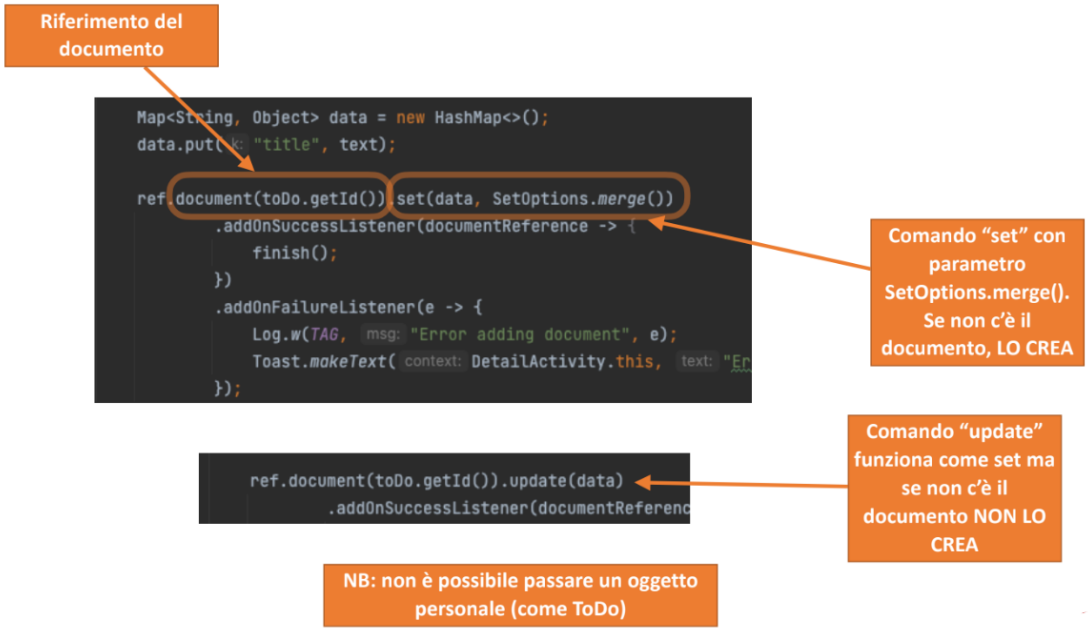
NB: fino a questo momento nulla è ancora avvenuto sul DB. Solo le operazioni successive andranno a creare la collezione SE NON ESISTE. Questo comportamento è tipico dei DB NoSQL

Azioni eseguibili sul DB

- Creazione di un documento

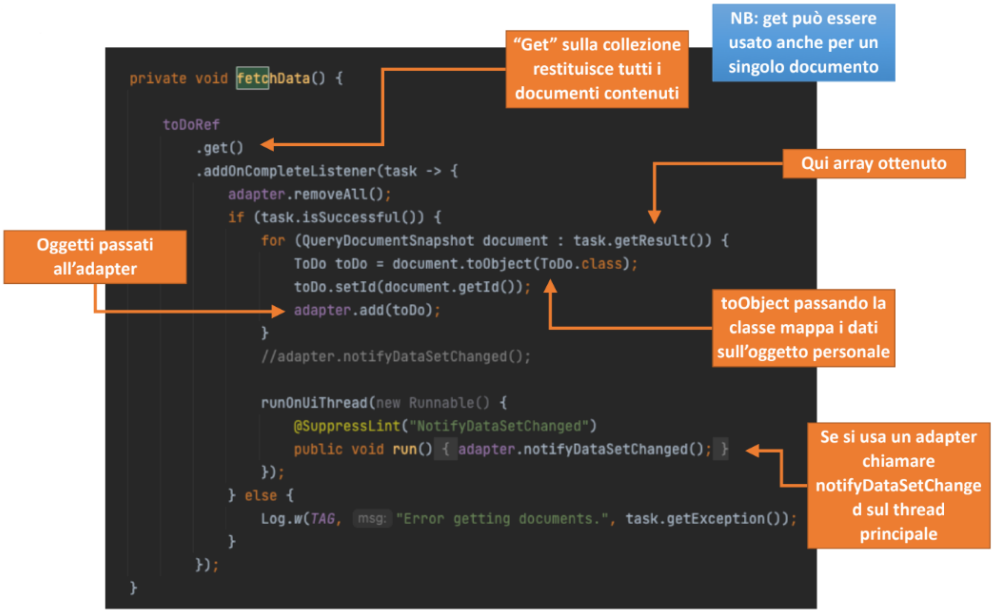


- Modifica del documento



- Cancellazione documento

- Lettura documenti



Libreria Picasso

lo scopo di qesta libreria esterna è quello di semplificare il caricamento di immagini remore e una gestione migliore della cache per evitare continuoi download. Per permettere il suo funzionamento è necessario importarla come dipendenza utilizzando poi il medo `load` per associarla all'immageView

nel file `Build.gradle` richiamo la libreria e la implemento

```
implementation 'com.squareup.picasso:picasso:2.8'
```

nel file java principale richiamo la libreiria e associo i diversi metodi

```
String imageUrl = "https://..."
Picasso.get() Picasso
    .load(imageUrl) RequestCreator
    .into(imageView, new com.squareup.picasso.Callback() {
```

```
        @Override
        public void onSuccess() {
            progressBar.setVisibility(View. INVISIBLE);
        }

        @Override
        public void onError(Exception e) {
            progressBar.setVisibility(View. INVISIBLE);
        }
});
```

nel file dello style XML posso creare una ProgressBra (spinner)

```
<ProgressBar
android:id="@+id/progressBar"
style="?android:attr/progressBarStyle"
android:Layout_width="wrap_content"
android:Layout_height="wrap_content"
android:Layout_marginTop="32dp"
app:Layout_constraintEnd_toEnd0f="parent"
app:Layout_constraintStart_toStartOf="parent"
app:Layout_constraintTop_toBottom0f="@+id/save_button" />
```