
UNIVERSITÀ
DI PARMA

# *Relazione Progetti di PROGRAMMAZIONE DI SISTEMI MOBILI*

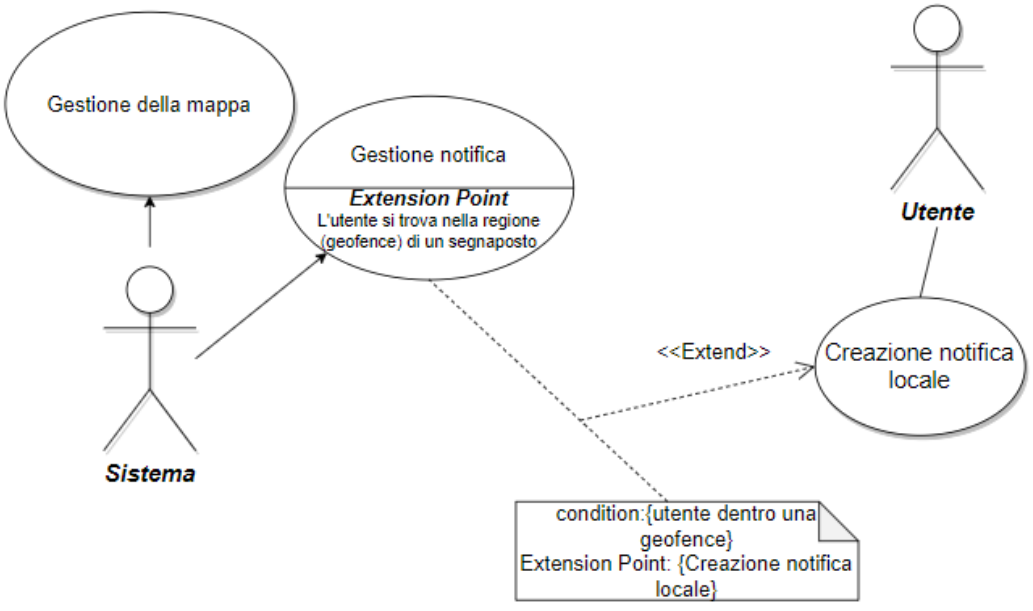
***Simone Pio Abate, 317249***

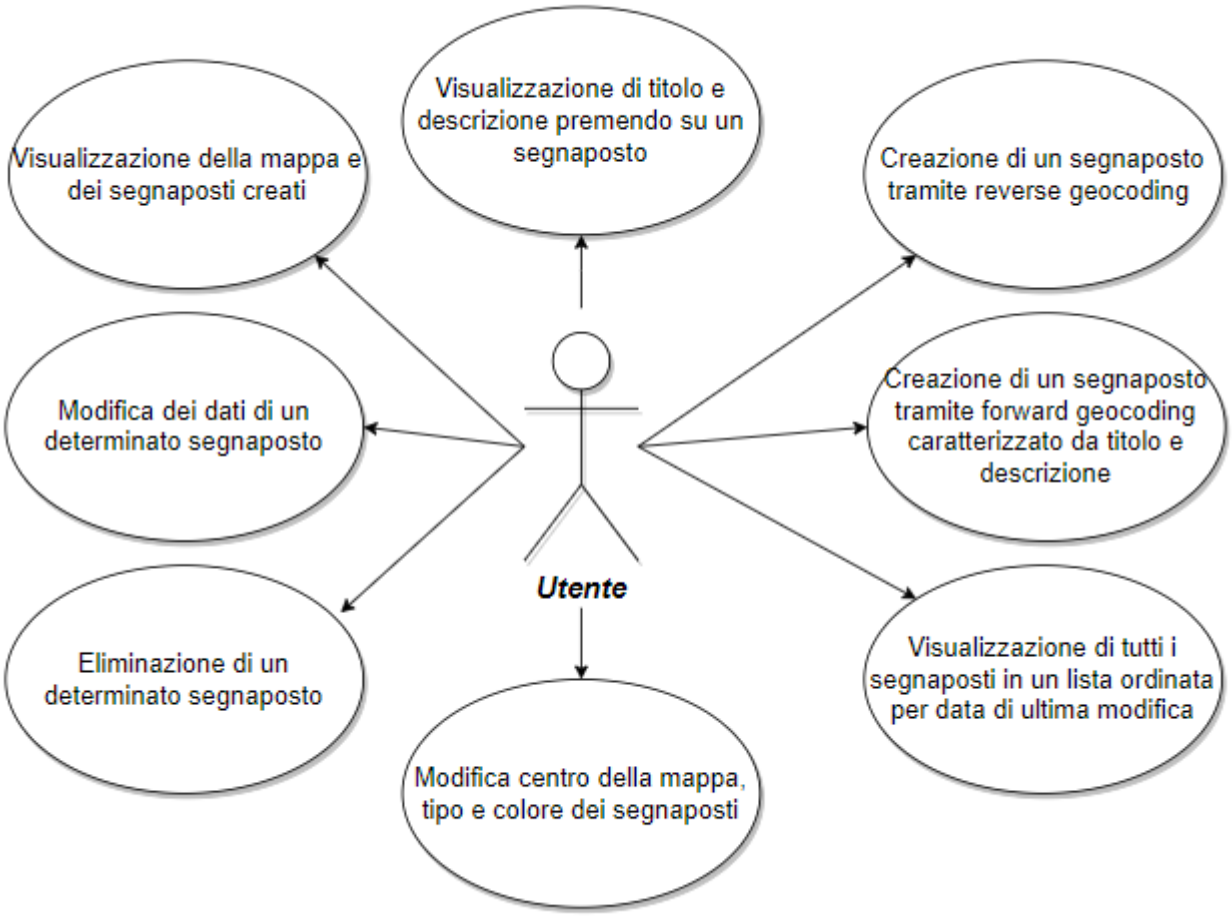
# PlaceReminder

PlaceReminder è un'applicazione mobile di mappatura geografica intuitiva e completa. Consente agli utenti di creare segnaposti personalizzati utilizzando sia il reverse geocoding, per aggiungere segnaposti basati sulla posizione attuale, sia il forward geocoding. Gli utenti possono esplorare la mappa e i segnaposti creati, visualizzare dettagli premendo su di essi e gestire facilmente i loro segnaposti attraverso una lista ordinata per data di ultima modifica. Inoltre, l'applicazione offre altre funzionalità come la modifica del centro della mappa, la tipologia della mappa e il colore dei segnaposti, oltre alla possibilità di ricevere notifiche locali quando si entra nella regione di un segnaposto specifico grazie alla funzione di geofencing.

## • CARATTERISTICHE PRINCIPALI:

1. **Creazione di un segnaposto tramite reverse geocoding :** L'utente può creare un segnaposto inserendo manualmente la longitudine e la longitudine oltre ad un titolo ed una descrizione.
2. **Creazione di un segnaposto tramite forward geocoding caratterizzato da titolo e descrizione:** L'utente può creare un premendo su un punto desiderato della mappa.
3. **Visualizzazione della mappa e dei segnaposti creati:** L'utente può visualizzare la mappa contenente tutti i segnaposti da esso inseriti
4. **Visualizzazione di titolo e descrizione premendo su un segnaposto:** Quando l'utente preme su un segnaposto viene mostrato il suo titolo e la sua descrizione, premendo ulteriormente su questo marker si avrà la possibilità di modificarne le caratteristiche.
5. **Visualizzazione di tutti i segnaposti in una lista ordinata per data di ultima modifica:** L'utente può visualizzare tutte le informazioni di tutti i segnaposti, i segnaposti sono ordinati in base alla data di ultima modifica.
6. **Modifica centro della mappa, tipo e colore dei segnaposti(impostazioni):** L'utente tramite le impostazioni può modificare le caratteristiche dell'applicazione, come il centro della mappa all'apertura di essa, la tipologia di mappa (normale, satellitare, terrestre, ibrida) e il colore dei segnaposti visualizzabili nella mappa.
7. **Eliminazione di un determinato segnaposto:** L'utente può eliminare i segnaposti.
8. **Modifica dei dati di un determinato segnaposto:** L'utente può modificare i dati di un segnaposto.
9. **Gestione della notifica:** Ogni segnaposto è caratterizzato da un sistema di Geofence, nel caso l'utente dovesse entrare nella sua regione, viene generata una notifica locale.





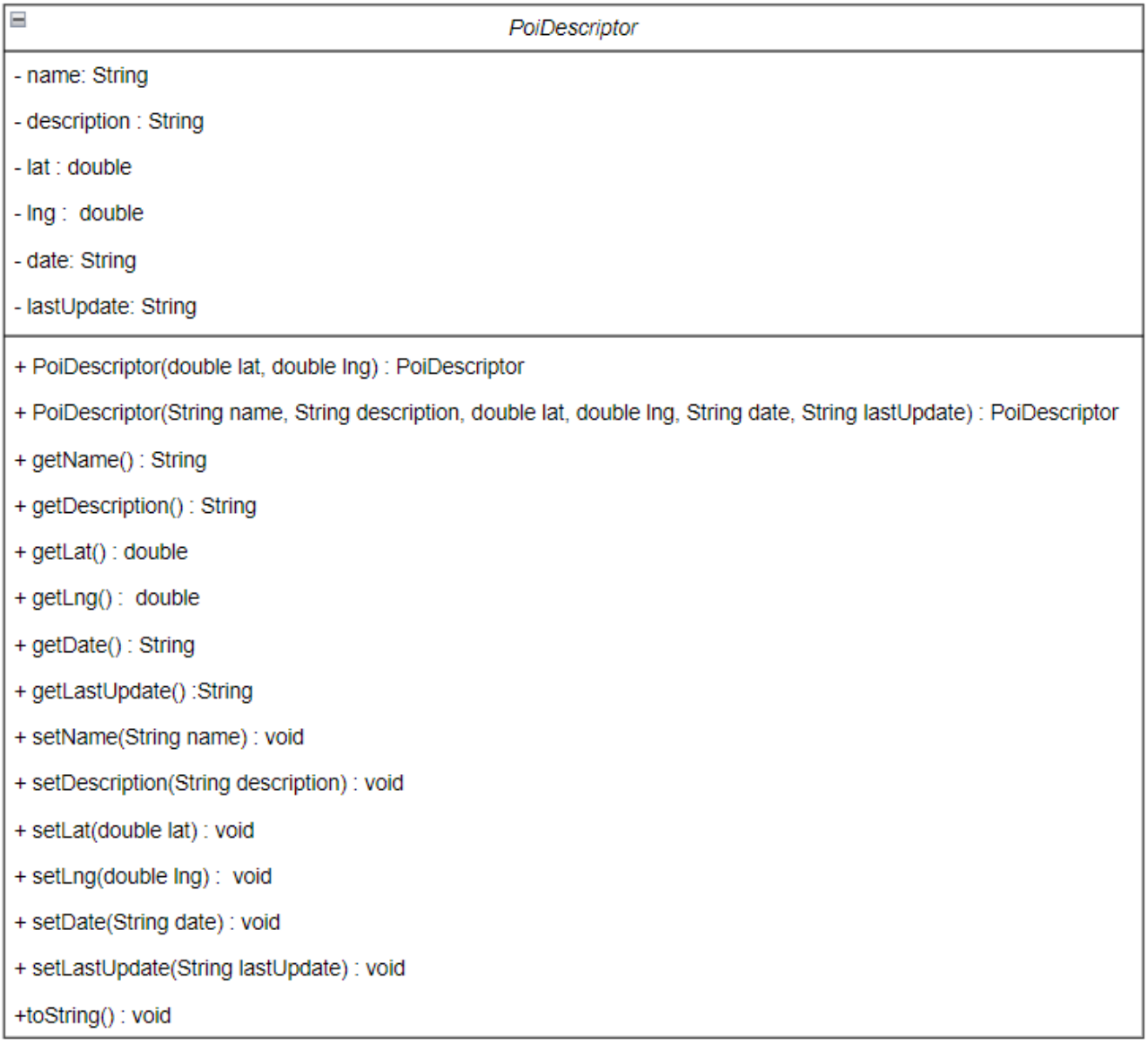

## • TECNOLOGIE USATE :

Il progetto myCurrency utilizzata le seguenti tecnologie:

- **Android Studio:** è la piattaforma principale su cui è stata sviluppata l'applicazione, utilizzando il Pattern MVC (Model View Controller) per garantire una struttura ben organizzata e mantenibile del codice. Il Model si occupa della gestione dei dati, il View si occupa della visualizzazione dell'interfaccia utente e il Controller gestisce le interazioni degli utenti e coordina il flusso di dati tra Model e View. Questo approccio aiuta a separare le diverse responsabilità dell'applicazione, facilitando lo sviluppo e il debugging.
- **Google Maps API:** è stata integrata nell'applicazione per gestire la visualizzazione della mappa. Questa API offre una vasta gamma di funzionalità, tra cui la possibilità di interagire con la mappa tramite tocco, ottenendo le coordinate geografiche (forward geocoding) in base al punto toccato. Questa funzionalità è stata fondamentale per consentire agli utenti di inserire nuovi segnaposti in posizioni specifiche sulla mappa con facilità e precisione.

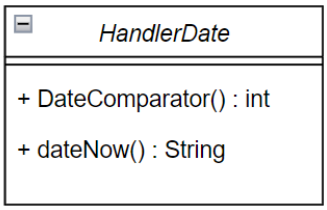
  Inoltre, Google Maps API offre anche funzionalità avanzate come la visualizzazione di marker personalizzati per i segnaposti, la gestione delle informazioni associate ai marker e la possibilità di calcolare percorsi e distanze. Tutte queste caratteristiche hanno contribuito a migliorare l'esperienza utente e la funzionalità complessiva dell'applicazione di mappatura geografica.

- **DIAGRAMMA DELLE CLASSI :**




La classe **PoiDescriptor** rappresenta la gestione dei segnaposti, con gli attributi sopra descritti, i relativi costruttori e i metodi get & set.

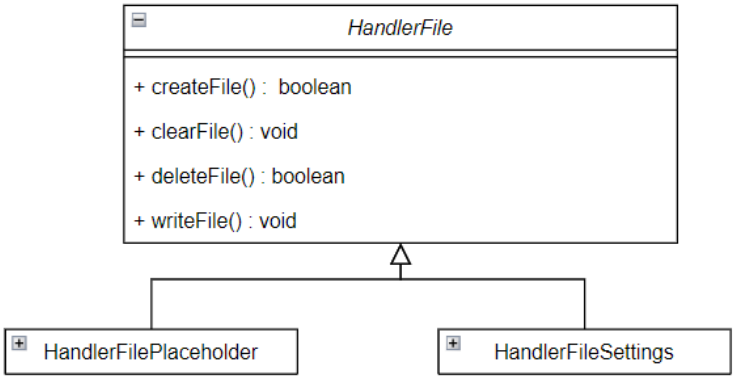
***toString():*** è utilizzato per stampare i dati del segnaposto come : “nome;descrizione;lat;lng;data creazione; data ultima modifica”, questo metodo è utilizzato nella gestione del segnaposto tramite il file "placeholders.txt".



La classe **HandlerDate** è stata utilizzata per la gestione delle date, anche se contiene poche funzionalità, ho ritenuto necessario l'implementazione di questa classe per dividere il più possibile le funzionalità dell'applicazione.

***DateComparator():*** Comparatore personalizzato che confronta due oggetti *PoiDescriptor* in base alla loro data di ultima modifica, confrontandole per ordine decrescente sulla data di ultima modifica.

***dateNow():*** restituisce la data di adesso *(dd-MM-yyyy HH:mm)* sotto forma di stringa.



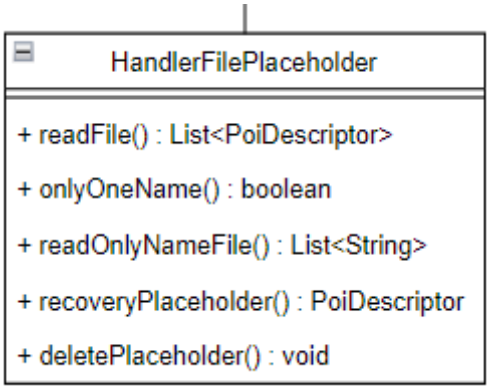

La classe **HandlerFile** è stata utilizzata per la gestione dei file, è una classe madre e non ci sono implementazioni nel codice di questa classe, ma è utilizzata solamente tramite *HandlerFIlePlaceholder* e *HandlerFileSettings*. Come le classi figlie tutti i metodi all'interno sono statici.

***createFile():*** utilizzata per creare i file, restituisce un valore booleano che nel caso il file esiste allora non farà nulla, altrimenti crea un file con valori di default (questo è utile soprattutto nel file delle impostazioni) .

***clearFile():*** utilizzata per pulire completamente un file, utilizzata per la riscrittura dei file.

***deleteFile():*** utilizzata per pulire eliminare un file, non è mai utilizzata nel codice ma ho preferito crearla lo stesso per eventualità future.

***writeFile():*** utilizzata per la scrittura di una linea nel file per quanto riguarda il file dei segnaposto e per la riscrittura completa per quanto riguarda quello delle impostazioni.




La classe **HandlerFilePlaceholder** è stata utilizzata per la gestione del file "placeholders.txt", che contiene tutte le informazioni riguardanti tutti i segnaposto

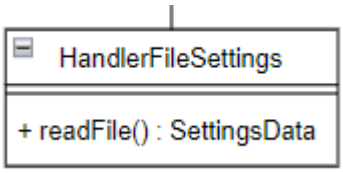
***readFile():*** funzionalità utilizzata per leggere il file e restituire una lista di tutti i PoiDescriptor.

***onlyOneName():*** Questa funzionalità si utilizza nel momento in cui si vuole aggiungere un nuovo segnaposto, difatti controlla se il nome desiderato per il segnaposto esiste già, restituendo appunto un valore booleano.

***readOnlyNameFile():*** utilizzata per recuperare tutti i nomi dei segnaposti ed aggiungerli al *AutoCompleteTextView*, per un accesso più rapido.

***recoveryPlaceholder():*** consente di recuperare il segnaposto tramite il solo nome, utilizzata quando si preme su un elemento del *AutoCompleteTextView* e anche nel menù delle impostazioni, nella scelta del segnaposto come centro della mappa.

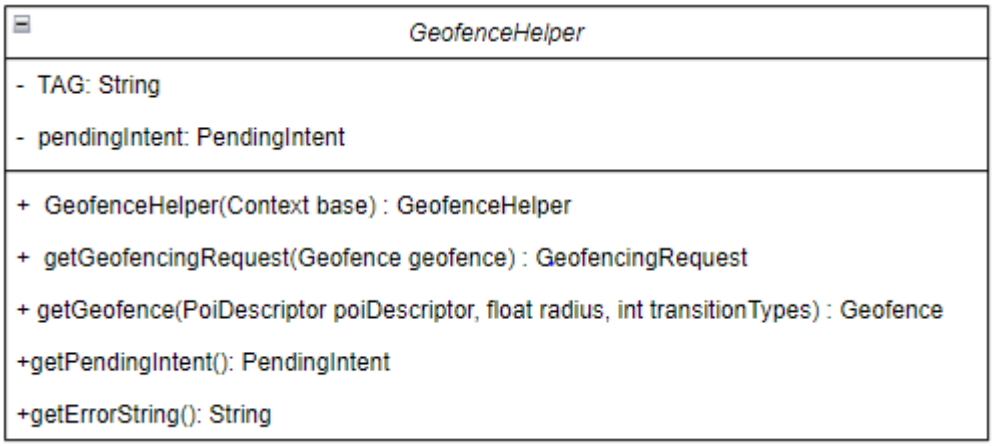
***deletePlaceholder():*** utilizzata per eliminare il segnaposto desiderato dal file.




***readFile() :*** consente di recuperare il file "settings.txt" e convertire i dati nella classe *SettingsData*

| *SettingsData* |
| --- |
| - latCenter : double<br>- lngCenter : double<br>- typeMap : String<br>- colorPlaceholder: String |
| + SettingsData(double latCenter, double lngCenter, String typeMap, String colorPlaceholder)<br>+ getLatCenter() : double<br>+ getLngCenter() : double<br>+ getTypeMap() : String<br>+ getColorPlaceholder() : String<br>+ setLatCenter(String latCenter) : double<br>+ setLngCenter(String lngCenter) : double<br>+ setTypeMap(String typeMap) : String<br>+ setColorPlaceholder(String colorPlaceholder) : void<br>+toString() : void |

La classe **SettingsData** rappresenta la gestione degli elementi del file "settings.txt", con gli attributi sopra descritti, il relativo costruttore e i metodi get & set. Questa classe è utilizzata solamente nella gestione del *SettingsFragment*.



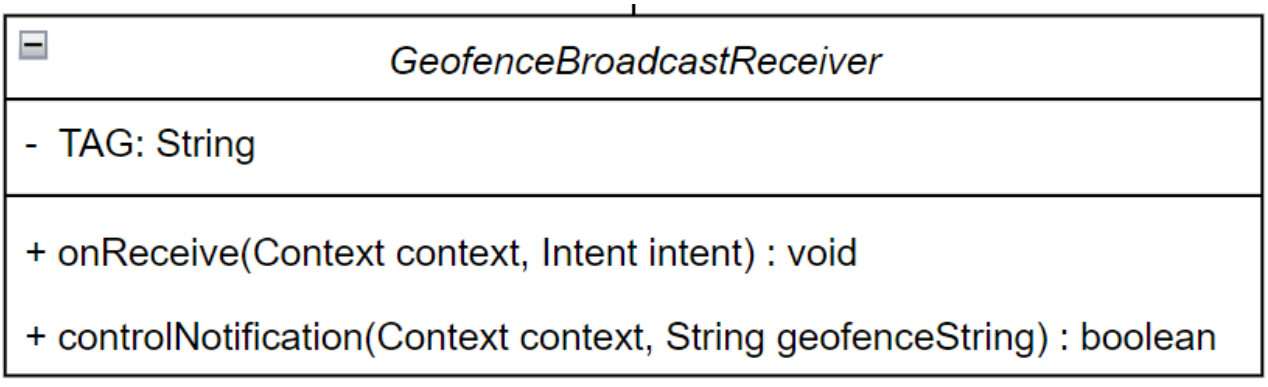

**GeofenceHelper** è la classe che gestisce la creazione del geofencing, è una classe di supporto che fornisce metodi per la creazione di richieste di geofence, oggetti di *geofence* e oggetti *PendingIntent* per la ricezione degli eventi di geofence. Classe utilizzata nel MapFragment nella creazione dei segnaposto

***getGeofencingRequest(Geofence geofence):*** utilizzato per ottenere una richiesta di geofence. La richiesta di geofence è un oggetto che rappresenta una richiesta al servizio di geofencing di Google Play Services per monitorare una determinata area geografica. In questo metodo, vengono impostati i parametri della richiesta di geofence, come la geofence stessa e il trigger iniziale, e infine viene restituita la richiesta completa.

***getGeofence(PoiDescriptor poiDescriptor, float radius, int transitionTypes)*** : utilizzata per creare e successivamente restituire un oggetto Geofence, passandolo poi in un secondo momento a *getGeofencingRequest()*.

***getPendingIntent():*** crea e restituisce l'oggetto PendingIntent che si occuperà di richiamare la classe *GeofenceBroadcast* quando richiesto.

***getErrorString():*** restituisce una stringa di errore in base al caso riscontrato, questa funzionalità non è utilizzata in nessuna parte del codice.

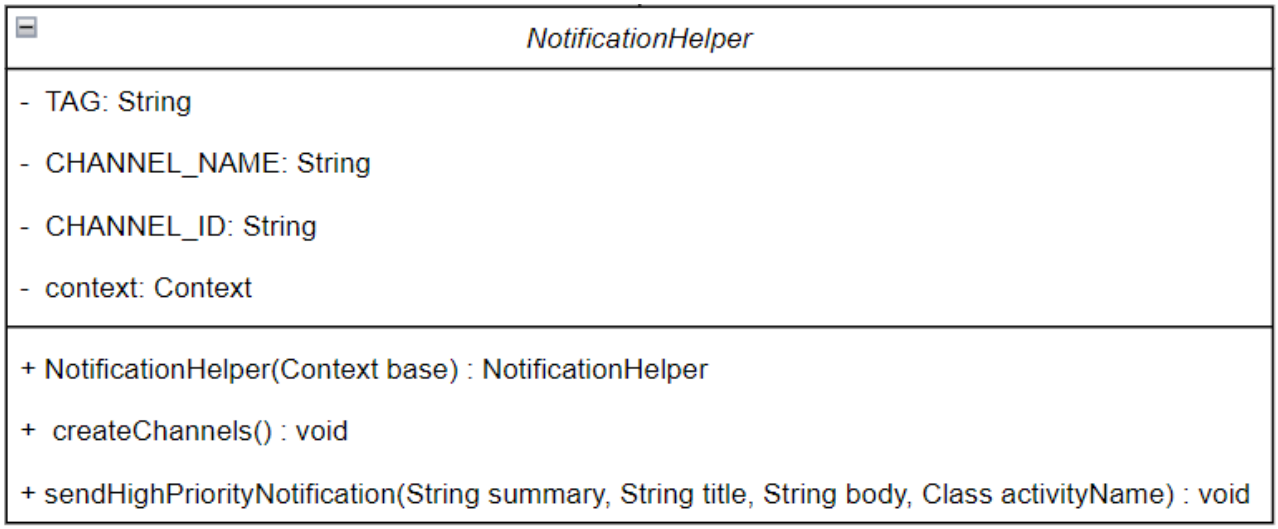



***GeofenceBroadcastReceiver*** viene richiamata quando viene scatenato il PendingIntent associato agli eventi di geofencing.

***onReceive(Context context, Intent intent):*** chiamato quando si verifica un evento di geofencing, tramite appunto il PendingIntent, questo metodo gestisce la creazione di un oggetto *NotificationHelper*.

***controlNotification(Context context, String geofenceString):*** viene utilizzato per controllare se una notifica identica è già attiva nel sistema. Questo metodo controlla le notifiche attive nel sistema e confronta il loro

contenuto con il messaggio della geofence. Restituisce true se non esiste una notifica identica attiva, altrimenti restituisce false.

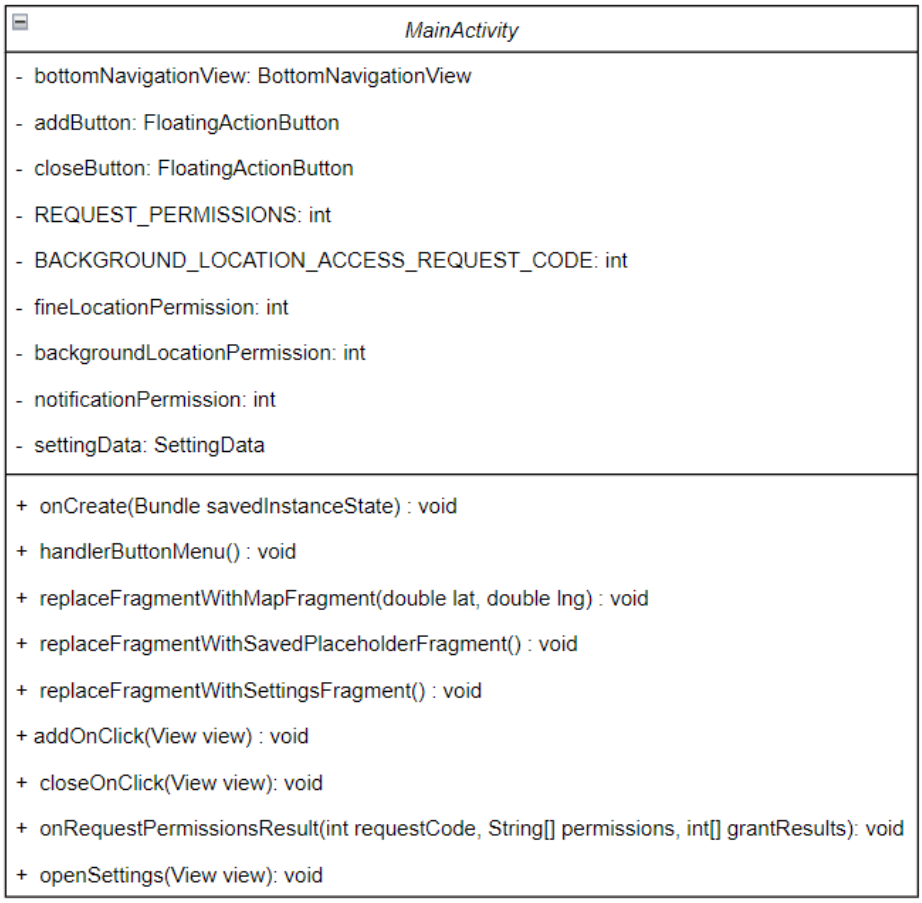



La classe ***NotificationHelper*** è progettata per semplificare la gestione delle notifiche all'interno di un'applicazione Android. Si occupa di creare e inviare notifiche ad alta priorità agli utenti.

***NotificationHelper(Context base):*** Nel costruttore, la classe controlla se il dispositivo è in esecuzione su una versione di Android uguale o superiore a Oreo (API 26). Se sì, crea un canale di notifica ad alta priorità utilizzando il metodo createChannels().

***createChannels():*** questo metodo si occupa di configurare e creare un canale di notifica ad alta priorità, necessario per garantire che le notifiche inviate tramite questo canale siano gestite correttamente anche sulle versioni più recenti di Android.

***sendHighPriorityNotification(String summary, String title, String body, Class activityName):*** questo metodo gestisce la creazione e l'invio di una notifica ad alta priorità.




***MainActivity*** è la classe che gestisce l'entrypoint dell'applicazione, difatti gestisce gli elementi grafici di base come:

***bottomNavigationView***: è fondamentalmente il menù dell'applicazione, gestendo l'interazione con i vari fragment ( *MapFragment*, *SavedPlaceholdersFragment* ,*SettingsFragment).*

***addButton:*** bottone che nel caso venga premuto farà scomparire il menù per dare la possibilità di aggiungere un segnaposto.

***closeButton:*** bottone che nel caso venga premuto farà riapparire il menù togliendo la possibilità di aggiungere un segnaposto.

Ed elementi di modelli come:

***settingData:*** utilizzato, dopo aver recuperato le informazioni del file, per impostare i valori scelti dall'utente e inseriti nel file.

Gli altri attributi sono utilizzati per la gestione dei permessi.

***onCreate(Bundle savedInstanceState):*** metodo fondamentale in ogni controller, difatti è colui che viene chiamato automaticamente all'apertura dell'applicazione, gestisce:

- La **creazione del file** con i valori di default (se necessario), oppure il **recupero dei dati** precedentemente inseriti, come tutti i segnaposti creati e i file delle impostazioni.
- La **gestione dei permessi**, quindi il chiedere i permessi se non ci sono e controllare la risposta dell'utente.
- Richiama al suo interno *handlerButtonMenu*() e la gestione dei listener di *addButton* e *closeButton*.

***handlerButtonMenu():*** implementa un listener per recuperare quale elemento del menù è stato premuto.

***replaceFragmentWithMapFragment(double lat, double lng):*** metodo utilizzato per aprire il controller della mappa (*MapFragment*), richiamato in caso di permessi concessi direttamente da *onCreate(Bundle savedInstanceState*) dove utilizzerà come *lat* e *lng* i dati recuperati da *settingData,* da *handlerButtonMenu*() nel caso si premesse l'elemento desiderato, e da altri controller inseriti successivamente per poter ricaricare la mappa all'aggiunta o modifica di un segnaposto.

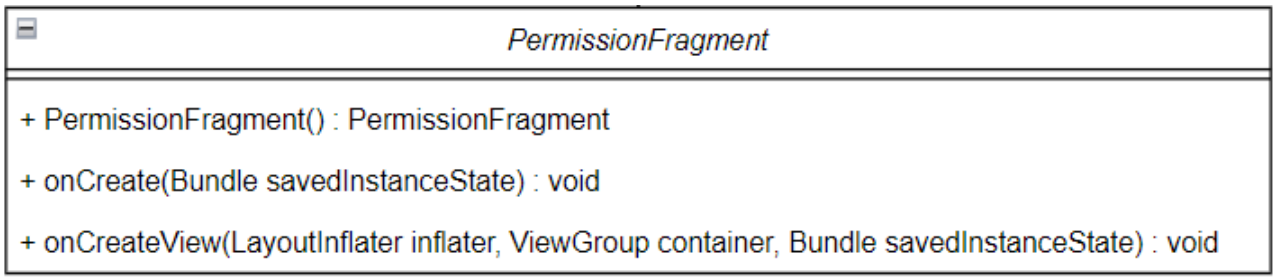
***replaceFragmentWithSavedPlaceholderFragment():*** utilizzato per aprire il controller della visualizzazione dei segnaposti creati (*SavedPlaceholderFragment*), richiamato da *handlerButtonMenu()* e anche nella ricarica del fragment alla modifica o eliminazione di un segnaposto.

***replaceFragmentWithSettingsFragment():*** utilizzato per aprire il controller che si occupa della gestione delle impostazioni (*SettingsFragment*), richiamato da *handlerButtonMenu(),* oppure nella riscrittura del file per ricaricare le modifiche.

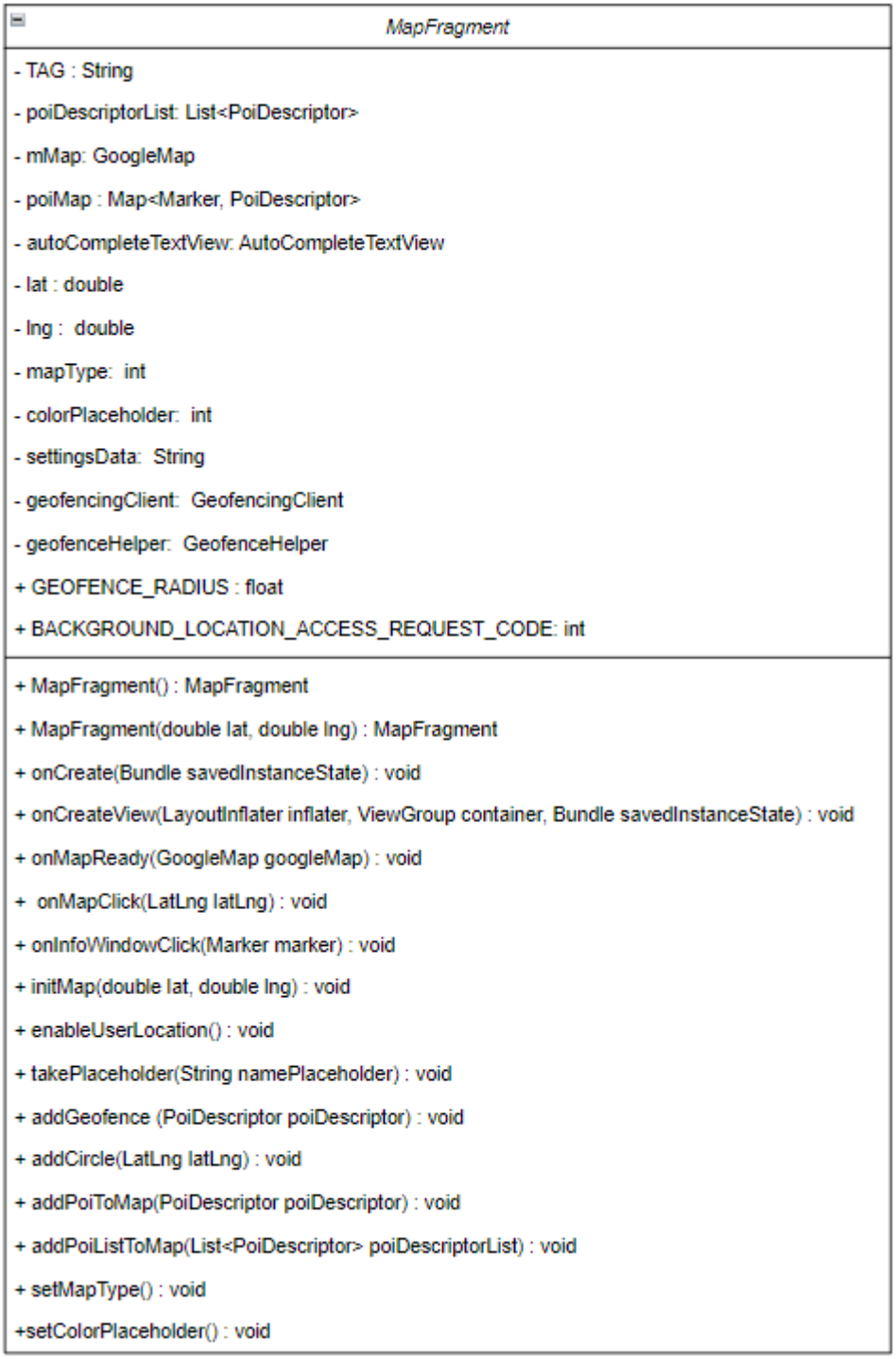
***addOnClick(View view):*** metodo utilizzato, come accennato, per dare la possibilità di aggiungere un segnaposto, in realtà non è proprio come, poiché l'aggiunta vera e propria è gestito nel *MapFragment*, difatti questo bottone serve solamente a rendere invisibile il menù e l'AutoCompleteTextView, il MapFragment notando questo cambiamento capirà che vogliamo aggiungere un segnaposto al click sulla mappa.

***closeOnClick(View view):*** metodo utilizzato, per rendere visibile il menù e l'AutoCompleteTextView, di conseguenza togliamo la possibilà di aggiungere un segnaposto.

***onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults):*** Il metodo onRequestPermissionsResult viene chiamato quando l'utente risponde a una richiesta di autorizzazione per l'uso delle autorizzazioni nel contesto dell'applicazione. Se l'utente accetta i permessi richiesti, allora verrà caricata la mappa, altrimenti verrà aperto il fragment *PermissionFragment ,* che tramite il metodo ***openSettings(View view),*** collegato al relativo bottone, gestito nel MainActivity, aprirà le impostazioni del telefono chiedendo appunto i permessi precedentemente rifiutati.




***PermissionFragment*** è un controller che, come unica vera e propria funzionalità, è quella di aprire le impostazioni tramite il bottone gestito nel **MainActivity.** Non ha nient'altro di rilevante.




**MapFragment** è la classe che gestisce principalmente la mappa recuperata tramite API da Google Maps, ma in realtà ha anche altre funzionalità come l'aggiunta di un segnaposto, la modifica e la gestione del Geofence di un segnaposto, assegnando ad ognuno di essi un receiver che genererà una notifica locale nel caso si entri nella regione definita.

***TAG:*** rappresenta il TAG del fragment, utile soprattutto per comunicare con i dialogFragment indicando che la richiesta è avvenuta da esso.

**poiDescriptorList:** contiene la lista dei segnaposti, recuperata tramite file e serve per crearli nella mappa.

**mMap**: rappresenta la mappa recuperata da Google Maps, molte funzionalità sono legate ad esse, come appunto la creazione dei segnaposti in quest'ultima

**poiMap**: serve per collegare i segnaposti della mappa(ossia i marker) ai PoiDescriptor, legandoli.

**autoCompleteTextView**: elemento utilizzato per la definizione delle parole contenute in esso, per accedere più rapidamente al segnaposto desiderato.

**geofencingClient:** contiene elementi necessari per accedere ai servizi di localizzazione e geofencing, utilizzata per assegnare il geofencing ai segnaposti.

**geofenceHelper:** oggetto utilizzato per l'aggiunta del geofence ai segnaposto con relativa gestione del trigger.

***onCreate(Bundle savedInstanceState):*** Gestisce solamente le dichiarazioni di *geofencingClient e geofenceHelper.*

***onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState):*** Gestisce la creazione della vista, recuperando la mappa, i segnaposti, aggiunge i suggerimenti all'*autoCompleteTextView*, recupera i valori delle impostazioni e le usa per richiamare *setMapType()* e *setColorPlaceholder()*.

***onMapReady(GoogleMap googleMap):*** viene chiamato in automatico quando la mappa è pronta per essere utilizzata. Imposta i valori di latitudine e longitudine e richiama il metodo *addPoiListToMap*().

***onMapClick(LatLng latLng):*** metodo richiamato quando l'utente preme su un punto della mappa, nel caso in cui l'*autoCompleteTextView* non sia visibile (ossia quando si preme il pulsante *addButton* del MainActivity aprirà *AddPlaceholderDialogFragment* che permetterà di aggiungere un segnaposto.

***onInfoWindowClick(Marker marker):*** quando si preme su una window di un segnaposto si aprirà un *AddOrEditPlaceholderReverseGeocodingDialogFragment*, avendo così la possibilità di modificarne i dati.

***initMap(double lat, double lng):*** metodo che inizializza i dati della mappa, impostandone il centro, il tipo della mappa, ed abilita la locazione dell'utente (*enableUserLocation*()). Dichiara inoltre i 2 eventi sopracitati ossia, *onMapClick() onInfoWindowClick().*

***enableUserLocation():*** controlla se sono stati consentiti i permessi di localizzazione e in caso positivo aggiunge la posizione dell'utente.

***takePlaceholder(String namePlaceholder):*** funzione richiamata tramite *l'autoCompleteTextView***,** consentendo di riaggiornare la mappa utilizzando come centro il segnaposto desiderato.

***addGeofence(PoiDescriptor poiDescriptor):*** aggiunge il geofence al poiDescriptor desiderato, questa funzione viene chiamata nell'aggiunta dei segnaposti nella mappa. Questa aggiunta è gestita principalmente tramite la classe *geofenceHelper*.

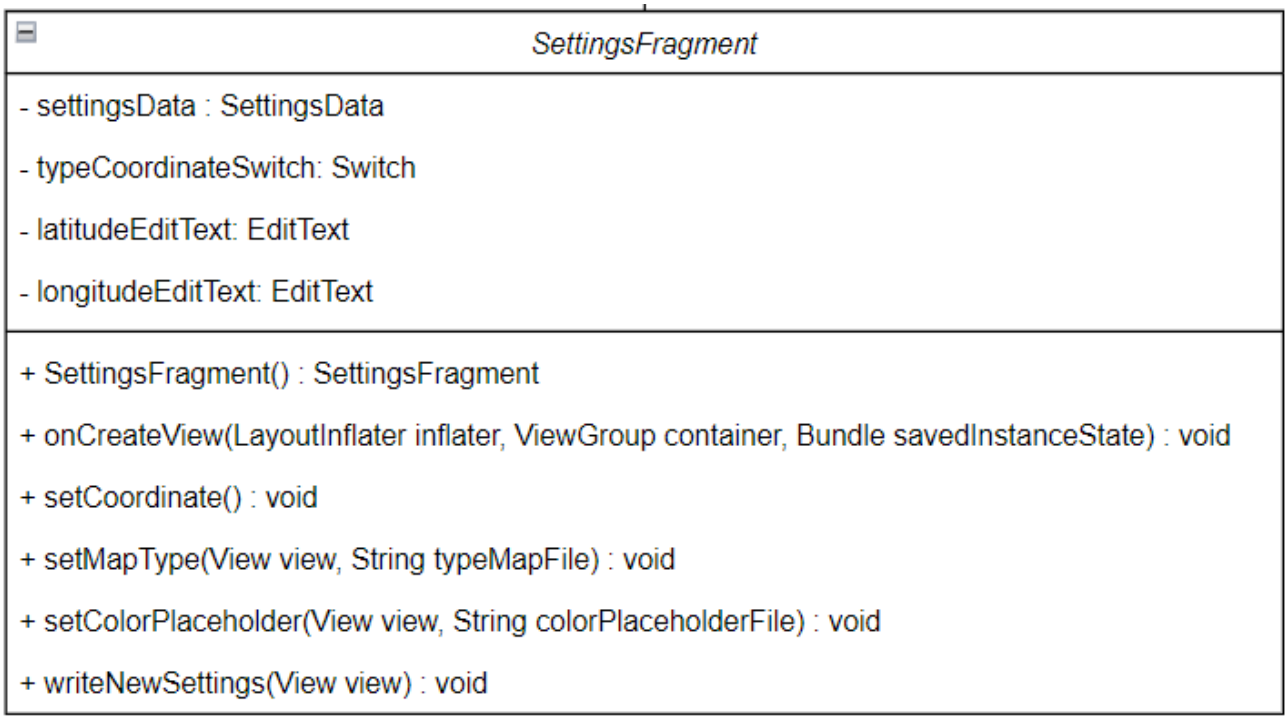
***addCircle(LatLng latLng):*** aggiunge un cerchio visibile intorno al segnaposto per determinare la zona di geofence.

***addPoiListToMap(List<PoiDescriptor> poiDescriptorList):*** richiama per ogni elemento della lista il metodo *addPoiToMap*().

***addPoiToMap(PoiDescriptor poiDescriptor):*** crea un oggetto MarkerOptions, inserendo le informazioni descritte nel poiDescriptor, richiamando anche i metodi *addGeofence()* e *addCircle().*

***setMapType():*** imposta il tipo di mappa, recuperato da "settings.txt".

***setColorPlaceholder ():*** imposta il colore dei segnaposti, recuperato da "settings.txt".

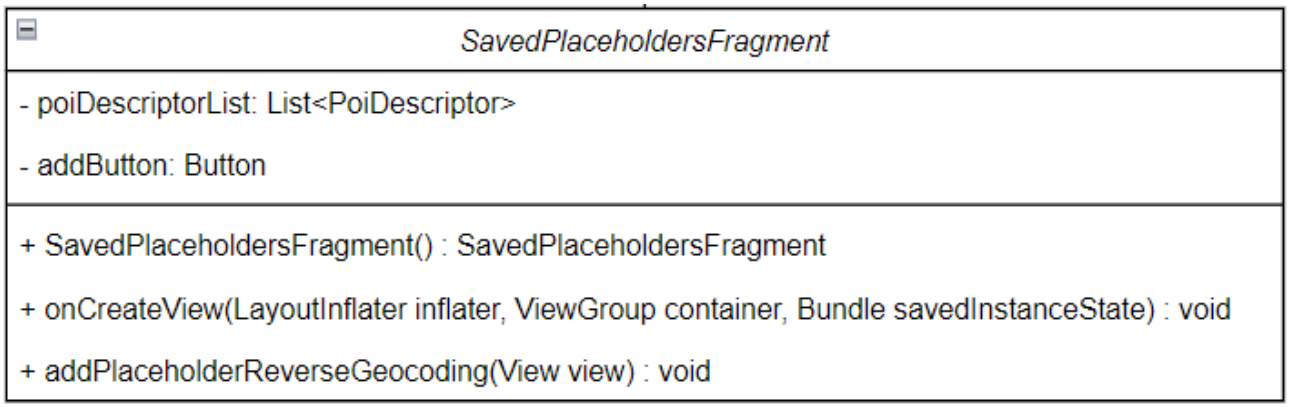



***SettingsFragment*** è il controller che gestisce le impostazioni, sovrascrivendole se l'utente lo desidera.

**settingData:** è l'oggetto che contiene di default gli elementi recuperati dal file "settings.txt"

***onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState):*** Questo metodo ha diverse funzionalità:

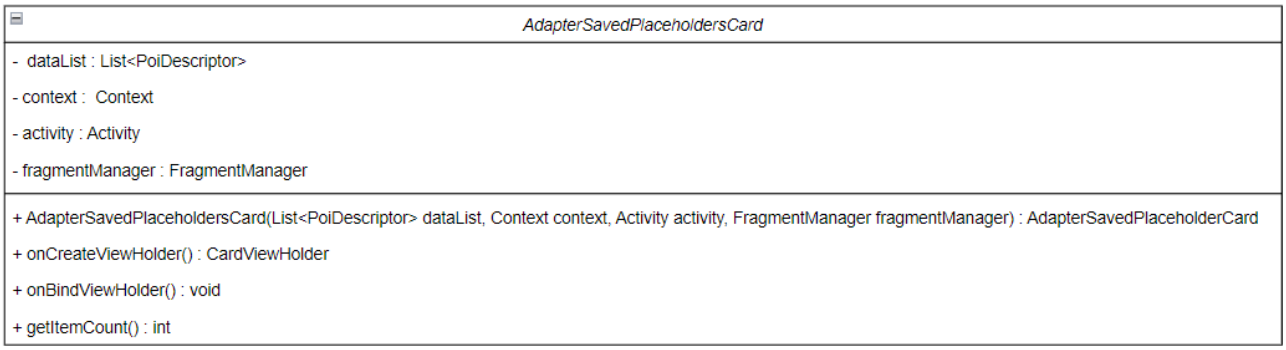
- recupera gli elementi dal file "settings.txt"
- Inizializza gli elementi dell'interfaccia utente come campi di testo per l'inserimento delle coordinate e *spinner* per la selezione dei segnaposti.
- Gestisce la selezione dei segnaposti, consentendo all'utente di visualizzare le coordinate associate al segnaposto selezionato.
- Consente all'utente di scegliere tra forward e reverse geocoding utilizzando uno switch.
- Imposta le coordinate (***setCoordinate()***), il tipo di mappa (***setMapType()***) e il colore del segnaposto (***setColorPlaceholder()***) in base alle impostazioni predefinite o salvate.
- Gestisce la selezione del tipo di mappa e del colore del segnaposto attraverso spinner dedicati.
- Gestisce l'evento di click sul pulsante per salvare le modifiche apportate alle impostazioni nel file "settings.txt". (***writeNewSettings()***)



***SavedPlaceholdersFragment*** è il controller che si occupa della visualizzazione di tutti i segnaposti, con la possibilità di poter modificare ed eliminare i segnaposti, oltre al poter crearne uno nuovo inserendo oltre al titolo e descrizione, la latitudine e la longitudine.

***onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState):*** crea un recyclerView, che richiama AdapterSavedPlaceholdersCard, dove verranno gestite per ogni segnaposto le funzionalità di modifica ed eliminazione.

***addPlaceholderReverseGeocoding(View view):*** metodo che gestisce l'evento che consente di aggiungere un segnaposto specificando titolo e descrizione, la latitudine e la longitudine.

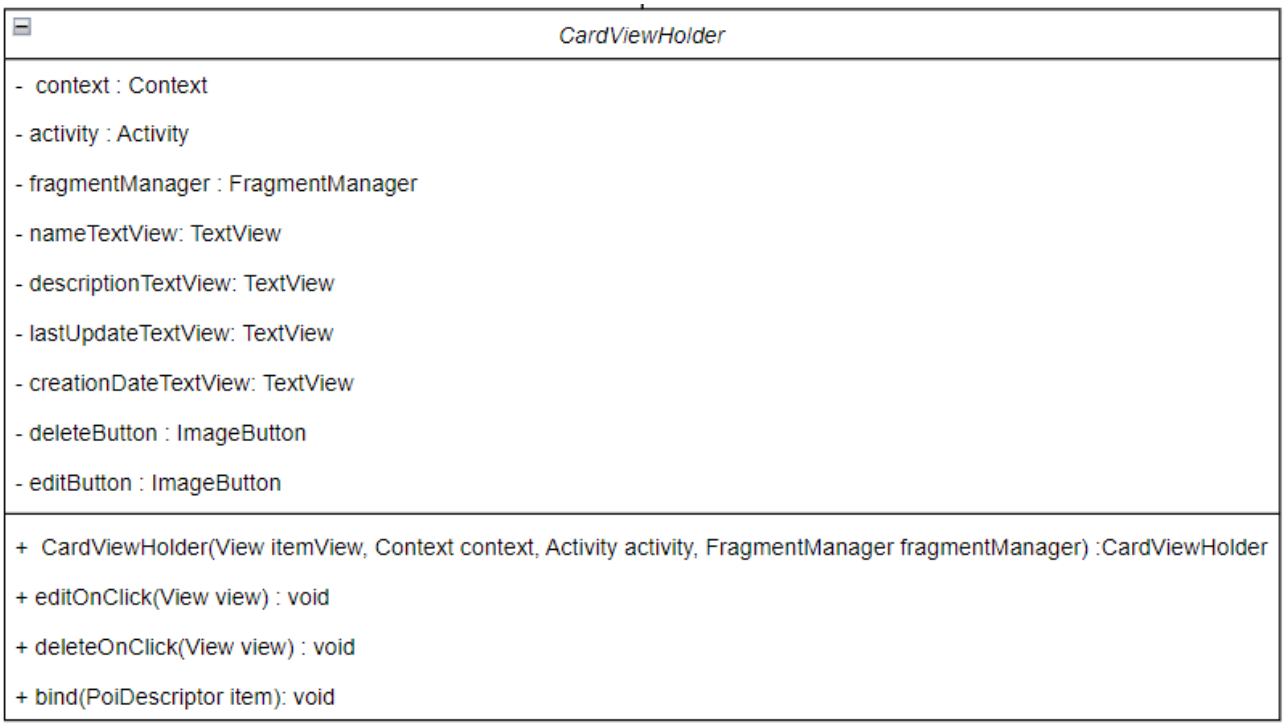



***AdapterSavedPlaceholdersCard*** estende *RecyclerView.Adapter<CardViewHolder>* e viene utilizzata per collegare i dati (nella lista poiDescriptorList) agli elementi della vista nella RecyclerView.

***onCreateViewHolder(ViewGroup parent, int viewType):*** Questo metodo viene chiamato quando la *RecyclerView* ha bisogno di creare una nuova istanza di *CardViewHolder*. Viene chiamato quando non esiste già una view da riutilizzare. All'interno di questo metodo viene restituita una nuova istanza di *CardViewHolder* associata a questa vista.

***onBindViewHolder(CardViewHolder holder, int position):*** Questo metodo viene chiamato quando la RecyclerView ha bisogno di collegare i dati ad una istanza di CardViewHolder. Viene chiamato ogni volta che un elemento della RecyclerView deve essere visualizzato o aggiornato. In questo metodo, si ottiene l'oggetto PoiDescriptor dalla lista poiDescriptorList in base alla posizione specificata, e quindi si chiama il metodo *bind* di *CardViewHolder* per associare i dati all'elemento della vista. (il PoiDescriptor nella posizione 0 della lista viene associato al CardViewHolder nella posizione 0 all'interno della RecyclerView e così via).

***getItemCount():*** restituisce il numero degli elementi presenti.



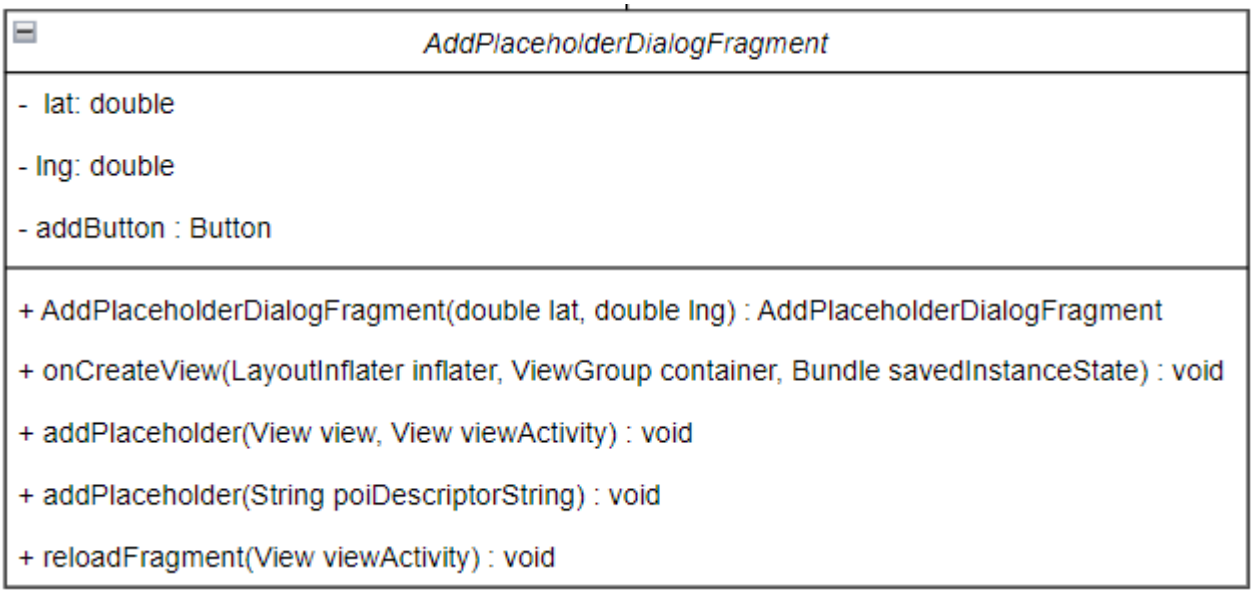

***CardViewHolder*** è la classe che si occupa di gestire singolarmente ogni elemento della lista (Card), difatti rappresenta un singolo elemento all'interno della RecyclerView. All'interno del costruttore, vengono inizializzati i vari elementi della vista utilizzando i loro ID, e vengono impostati i listener per i pulsanti di modifica ed eliminazione .

***CardViewHolder(View itemView, Context context, Activity activity, FragmentManager fragmentManager):*** Inizializza le variabili di istanza context, activity e fragmentManager e si occupa di collegare i componenti UI (TextViews, ImageButtons) alle rispettive variabili di istanza utilizzando gli ID del file xml.

***editOnClick(View view):*** dopo aver recuperato il segnaposto in questione apre *AddOrEditPlaceholderReverseGeocodingDialogFragment* per poter modificarne le informazioni.

***deleteOnClick(View view):*** dopo aver recuperato il segnaposto in questione, apre un *AlertDialog* con la possibilità di poter eliminarlo.

***bind(PoiDescriptor item):*** imposta le informazioni recuperate nell'*AdapterSavedPlaceholdersCard*.




**AddPlaceholderDialogFragment** è un *DialogFragment* che consente all'utente di aggiungere un nuovo segnaposto sulla mappa.

***AddPlaceholderDialogFragment():*** Inizializza le variabili di istanza lat e lng con quelle recuperate al click nella mappa.

***addPlaceholder(View view, View viewActivity):*** Recupera il nome e la descrizione del segnaposto inseriti dall'utente, verifica che entrambi i campi siano stati compilati e che il nome del segnaposto sia unico (tramite *onlyOneName()* della classe *HandlerFilePlaceholder*. Se i dati sono validi, crea un nuovo oggetto *PoiDescriptor* con i dati inseriti e lo scrive nel file "placeholders.txt". Infine, chiama il metodo *reloadFragment* per ricaricare la mappa e il menu dell'attività principale.

***addPlaceholder(String poiDescriptorString):*** aggiunge il segnaposto al file e chiude il *DialogFragment*.

***reloadFragment(View viewActivity):*** ricarica il MapFragment passando come latitudine e longitudine quelle del segnaposto appena creato.


***AddOrEditPlaceholderReverseGeocodingDialogFragment*** è un DialogFragment che consente all'utente di aggiungere o modificare un segnaposto utilizzando il segnaposto passato come argomento.

***AddOrEditPlaceholderReverseGeocodingDialogFragment():*** costruttore utilizzato nel caso si volesse creare il segnaposto, notiamo che è diverso da *AddPlaceholderDialogFragment* poiché si deve gestire due EditText in più per aggiungere la latitudine e la longitudine.

***AddOrEditPlaceholderReverseGeocodingDialogFragment(PoiDescriptor poiDescriptor):*** costruttore utilizzato nel caso si volesse modificare il segnaposto, passandolo appunto come argomento.

***onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState):*** si occupa di collegare i componenti UI (TextViews, ImageButtons) alle rispettive variabili di istanza utilizzando gli ID del file xml. Nel caso il poiDescriptor non sia nulla (ossia se è stato passato nel costruttore) riempie i campi con i suddetti valori.

***handlerPlaceholder():*** metodo richiamato nel caso si premesse sul bottone *addOrEditButton,* gestendo l'inserimento del segnaposto (***addPlaceholder()***) oppure la modifica di esso (***editPlaceholder()***).

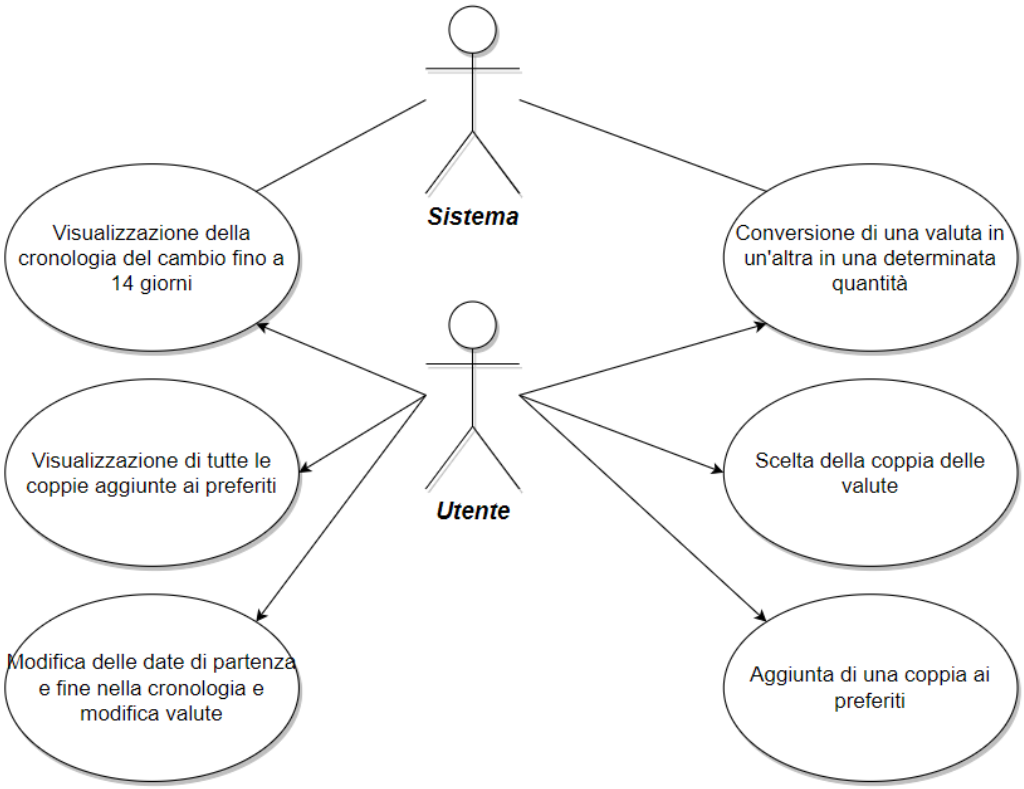
***reloadFragment(View viewActivity):*** ricarica il MapFragment passando come latitudine e longitudine quelle del segnaposto appena creato/modificato, oppure il SavedPlaceholderFragment dipendentemente dal TAG.

# MyCurrency

MyCurrency è un'applicazione mobile progettata per semplificare la conversione di valute in modo rapido e intuitivo. Grazie alle sue caratteristiche, gli utenti possono gestire facilmente le proprie esigenze di cambio valuta in qualsiasi momento.

- **CARATTERISTICHE PRINCIPALI:**

1. **Conversione Valutaria Precisa:** L'utente può convertire una quantità specifica di una valuta in un'altra con precisione e facilità.
2. **Scelta delle Coppie Valutarie:** L'utente può di selezionare liberamente le coppie di valute desiderate per la conversione, offrendo flessibilità e personalizzazione.
3. **Gestione delle Preferenze:** L'utente ha la possibilità di aggiungere le sue coppie di valute preferite a un elenco dedicato, rendendo più veloce e accessibile la conversione delle valute più frequentemente utilizzate.
4. **Personalizzazione della Cronologia:** L'utente può modificare le date di inizio e fine nella cronologia delle conversioni, oltre a poter modificare le valute coinvolte in ciascuna transazione.
5. **Visualizzazione delle Coppie Preferite:** L'utente può visualizzare facilmente tutte le coppie di valute aggiunte ai preferiti, consentendo un accesso rapido alle conversioni preferite.
6. **Storico delle Conversioni:** L'utente ha la possibilità di visualizzare la cronologia del cambio valutario fino a 14 giorni, permettendogli di monitorare e analizzare le fluttuazioni delle valute nel tempo.



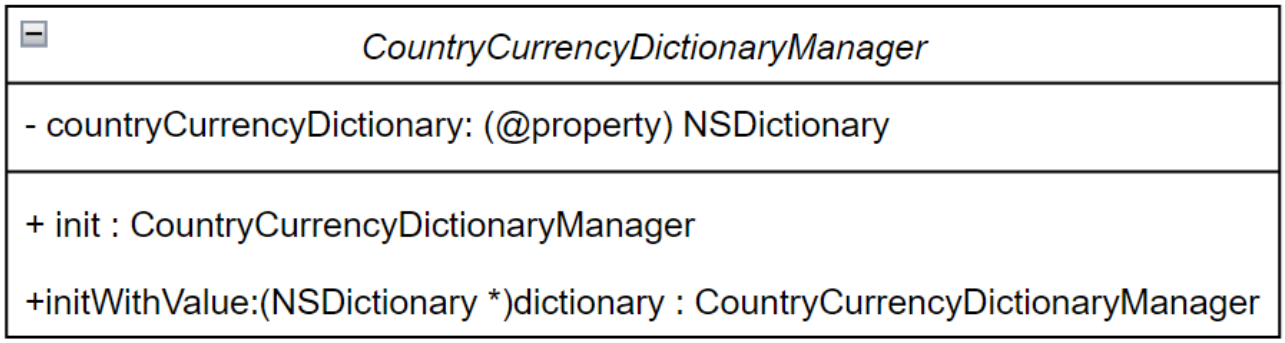

## • TECNOLOGIE USATE :

- **Xcode (Objective-C):** Xcode è l'ambiente di sviluppo integrato (IDE) ufficiale per la creazione di app iOS e MacOS. Nel contesto di "myCurrency", Xcode viene utilizzato per creare, sviluppare e testare l'applicazione, offrendo una vasta gamma di strumenti e funzionalità per semplificare il processo di sviluppo. Anche qui si utilizza il Pattern MVC (Model View Controller) per garantire una struttura ben organizzata e mantenibile del codice.
- **api FastFOREX:** servizio che fornisce dati aggiornati e affidabili sulle tariffe di cambio valuta in tempo reale. Grazie a questa API, "MyCurrency" può ottenere informazioni cruciali sulle

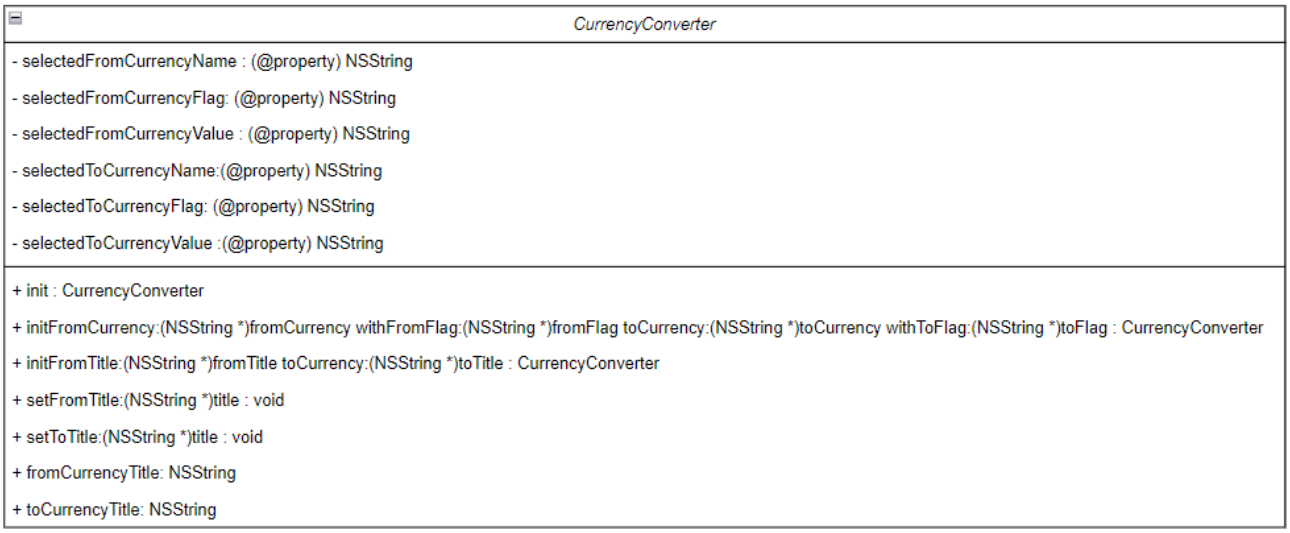
valute e utilizzarle per eseguire conversioni precise e accurate all'interno dell'applicazione. Alcune delle caratteristiche principali dell'API FastFOREX includono:

- ***Dati aggiornati in tempo reale:*** L'API fornisce dati sulle tariffe di cambio valuta che sono costantemente aggiornati in tempo reale, garantendo che le informazioni utilizzate dall'applicazione siano sempre attuali e accurate.
- ***Ampia copertura delle valute:*** FastFOREX offre una vasta gamma di valute supportate, consentendo agli utenti di convertire praticamente qualsiasi valuta in un'altra con facilità.
- ***Affidabilità e prestazioni:*** L'API FastFOREX è progettata per essere affidabile e performante, garantendo tempi di risposta rapidi e prestazioni elevate anche in situazioni di carico elevato.
- ***Facilità d'uso:*** L'API è progettata con un'interfaccia semplice e intuitiva facilitandone l'integrazione.

## • DIAGRAMMA DELLE CLASSI :

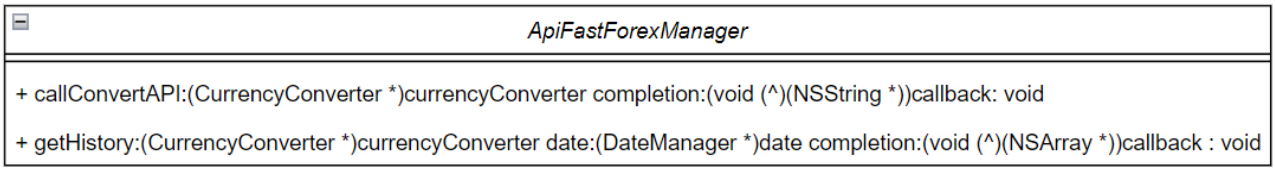



***CountryCurrencyDictionaryManager*** è la classe che si occupa della creazione del dizionario di tutte le valute, associando al nome (per esempio, EUR) la relativa bandiera, il metodo ***init*** chiama a sua volta il metodo ***initWithValue*** passandogli come argomento un dizionario, per la creazione dell'oggetto.




***CurrencyConverter*** è il modello principale, si occupa principalmente di salvare i dati delle coppie selezionate, come appunto il nome, bandiera e quantità, in più è disponibile la stringa *CurrencyTitle* per mostrare la coppia come "nome:bandiera", con la possibilità di creare l'oggetto proprio a partire da questa

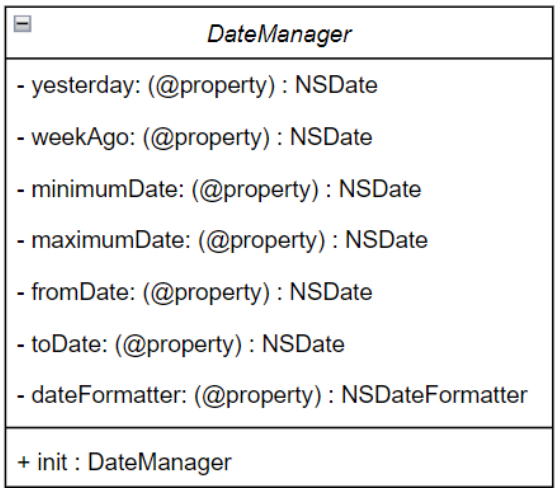
stringa, oppure partendo dai dati singoli. Una piccola osservazione, avendo creato gli oggetti tramite la ***@property*** i metodi get e set sono creati in automatico, senza il dover crearli manualmente. Ovviamente è possibile riscrivere queste funzionalità, come nel caso di ***setFromTitle*** e ***setToTitle,*** utilizzati nel già citato ***initFromTitle***, per separare i nomi dalle bandiere.




***ApiFastForexManager*** si occupa di creare e gestire le richieste all'API FastFOREX.

***callConvertAPI:(CurrencyConverter *)currencyConverter completion:(void (^)(NSString *))callback:*** è un metodo che crea una richiesta per convertire valute, prendendo in considerazione gli errori e le risposte ricevute. Se si verifica un errore durante l'invio della richiesta o nel recupero della risposta, il metodo lo registra e restituisce nil al blocco di completamento. In caso di risposta valida, il metodo analizza i dati ricevuti, estrae il tasso di conversione e lo utilizza per calcolare il risultato della conversione. Questo risultato viene quindi passato al blocco di completamento (*callback*).

***getHistory:(CurrencyConverter *)currencyConverter date:(DateManager *)date completion:(void (^)(NSArray *))callback:*** questo metodo recupera la cronologia delle conversioni valutarie per un intervallo di date specificato. Gestisce gli errori e le risposte ricevute durante l'invio della richiesta HTTP. In caso di risposta valida, analizza i dati ricevuti, ordina le date in modo ascendente e aggiorna i dati della tabella con le date ordinate e i corrispondenti valori di conversione. Infine, passa il risultato al blocco di completamento.



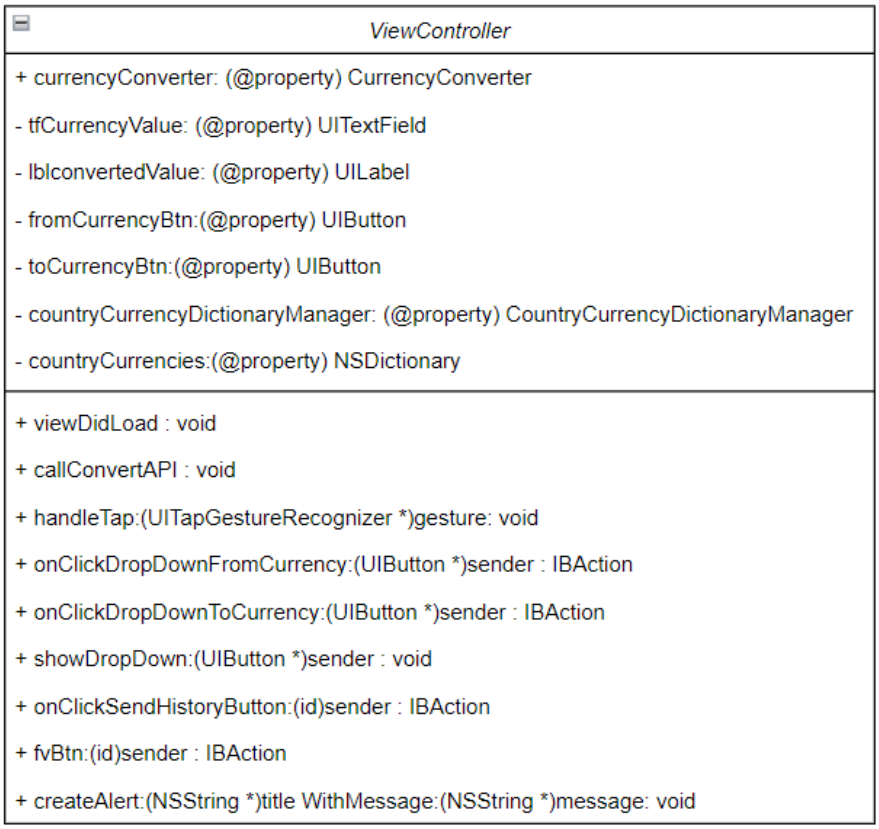

La classe ***DateManager*** gestisce le date, difatti creandone un oggetto, ossia richiamando l'***init***, impostiamo le date sopra elencate, utili soprattutto nella gestione della richiesta API ***getHistory***. Dove:

***yesterday*** e ***weekAgo,*** rappresentano rispettivamente, la data di ieri e di una settimana fa, utilizzate per inserirle nella richiesta.

***minimumDate*** e ***maximumDate***, rappresentano le date utilizzate per impostare la data minima e massima nei *DatePicker*.

***dateFormatter*** è utilizzata per formattare le date nel formato : *"yyyy-MM-dd"*

***fromDate*** e ***toDate*** sono quelle delle due valute, per quanto possa sembrare una ridondanza, in questo modo è possibile inviare un singolo oggetto a ***getHistory*** di ***ApiFastForexManager***.




***ViewController*** è la classe che gestisce l'entrypoint dell'applicazione, contenendo i seguenti elementi e i seguenti metodi:

***tfCurrencyValue:*** è la *UITextField* che si occupa della gestione della quantità della valuta di partenza.

***lblconvertedValue***: è la UILabel dove verrà visualizzato la quantità della valuta convertita.

***fromCurrencyBtn***: è un UIButton, utilizzato per cambiare il titolo della valuta di partenza.

***toCurrencyBtn***: è un UIButton, utilizzato per cambiare il titolo della valuta di destinazione.

***currencyConverter:*** è un oggetto *CurrencyConverter*, contiene, come detto in precedenza, tutti i dati riguardanti alle 2 valute.

***countryCurrencyDictionaryManager*** : è un oggetto della classe *CountryCurrencyDictionaryManager,* contenente tutti i titoli delle valute.

***viewDidLoad***: è la classe che gestisce l'entrypoint dell'applicazione, impostando i valori delle valute, prendendone i valori di default o quelli passati, nel caso, dal controller *FavViewController*, chiama la funzione ***callConvertAPI*** che a sua volta gestirà la richiesta svolta tramite la classe *ApiFastForexManager.*

***callConvertAPI:*** metodo utilizzato per comunicare con la classe *ApiFastForexManager* per gestire la richiesta.

***handleTap:(UITapGestureRecognizer *)gesture:*** questo metodo gestisce un qualsiasi tocco nel controller, difatti quando si tocca il controller, a meno che, non si premano altri bottoni oppure la tastiera per cambiare il campo *tfCurrencyValue*, fa partire il metodo ***callConvertAPI***.

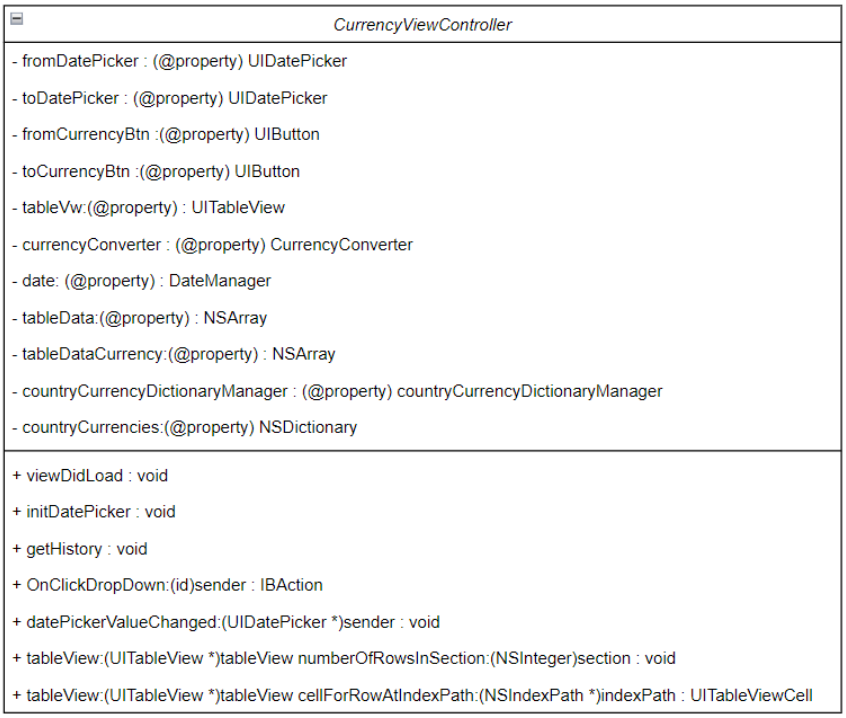
***onClickDropDownFromCurrency*** e ***onClickDropDownToCurrency:*** metodi che gestiscono i 2 bottoni utilizzati per cambiare i titoli delle 2 valute, e di conseguenza i valori di *currencyConverter*.

***showDropDown***: *onClickDropDownFromCurrency* e *onClickDropDownToCurrency,* chiamano questo metodo, che come accennato, gestisce il cambio del titolo del bottone e i valori del *currencyConverter,* in realtà crea un elemento per ogni componente di *countryCurrencyDictionaryManager,* inserendoli in un menù, dove alla scelta di un componente cambierà i valori della valuta di partenza o destinazione dipendentemente dall' ***onClickDropDown*** scelto. Infine, chiamerà ***callConvertAPI.***

***onClickSendHistoryButton:*** apre il controller ***CurrencyViewController***, passando come argomento *currencyConverter*.

***fvBtn***: quando si preme il bottone collegato a questo metodo, cercherà di aggiungere la coppia attuale di valute nei preferiti, controllando che non ne faccia già parte. L'aggiunta viene fatta attraverso la classe ***NSUserDefaults*** è una classe utilizzata per memorizzare le preferenze dell'utente, difatti memorizza i dati nel "defaults system", una sorta di database chiave-valore, che è persistente attraverso riavvii dell'applicazione o del dispositivo ,infine chiama ***createAlert.***

***createAlert***: metodo utilizzato per creare un semplice alert che mostrerà all'utente se la coppia è stata aggiunta con successo, oppure se già esisteva.




***CurrencyViewController*** gestisce la visualizzazione dei dati relativi alla cronologia della conversione valutaria, utilizzando ***UIDatePicker*** per selezionare l'intervallo di date desiderato e due *pulsanti* per selezionare le valute di origine e di destinazione. I dati della tabella sono ottenuti attraverso una richiesta API tramite la classe ***ApiFastForexManager***, e vengono visualizzati in una ***UITableView***. La classe utilizza anche una ***CountryCurrencyDictionaryManager*** per gestire le coppie valuta-paese e un DateManager per gestire le date.

***viewDidLoad:*** è chiamato quando si preme il bottone riguardante la cronologia nel ***ViewController***. In questo metodo:

- Vengono impostati i valori predefiniti per i pulsanti di selezione valuta utilizzando i titoli delle valute ottenuti dall'oggetto currencyConverter, passato alla attraverso il metodo ***onClickSendHistoryButton*** di ViewController.

- Viene inizializzato il dizionario ***countryCurrencies*** con le coppie valuta-paese ottenute dall'istanza di ***CountryCurrencyDictionaryManager***.
- Vengono inizializzati i DatePicker e l'oggetto ***date*** della classe ***DateManager*** utilizzando il metodo ***initDatePicker***.
- Viene chiamato il metodo ***getHistory*** per recuperare la cronologia delle conversioni valutarie.

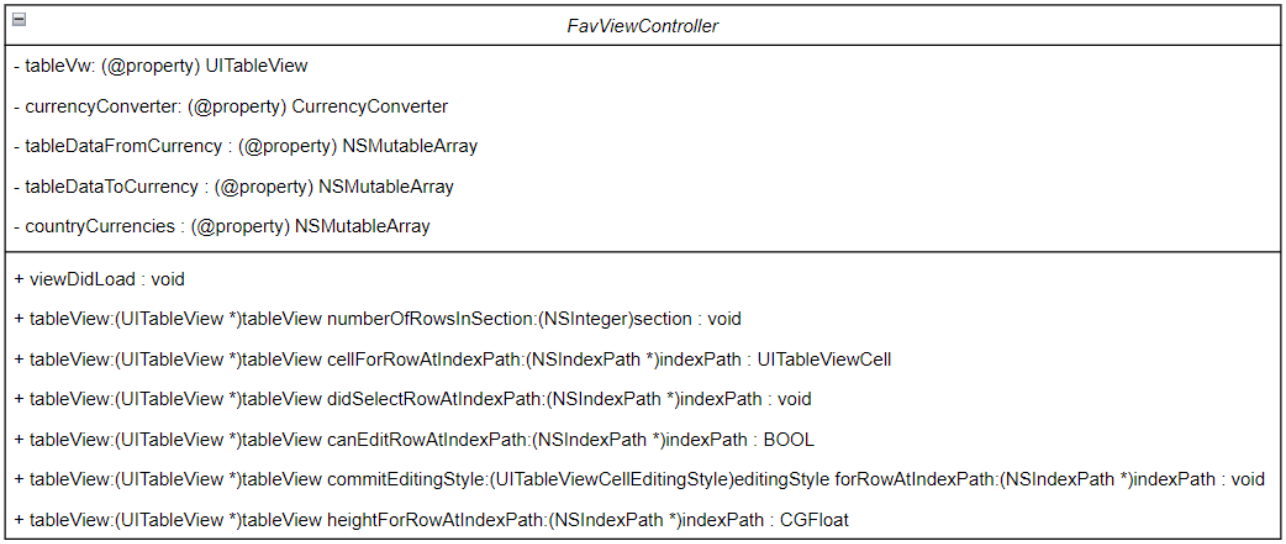
***getHistory:*** metodo utilizzato per comunicare con la classe *ApiFastForexManager* per gestire la richiesta della cronologia di una determina coppia con le date inserite nei DatePicker.

***OnClickDropDown***: stessa funzionalità del ViewController.

***datePickerValueChanged***: cambia il valore del DatePicker selezionato con il valore scelto.

***tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section:*** Restituisce il numero di righe nella tabella basato sulla lunghezza dei dati della tabella.

***(UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath:*** Metodo per creare e restituire una cella per la tabella, impostandone il testo della cella con la valuta e il tasso di cambio corrispondenti




***FavViewController*** gestisce le valute preferite dell'utente. Recupera le preferenze dell'utente da ***NSUserDefaults***, inizializza la tabella e imposta i delegati per gestire gli eventi. La tabella mostra le coppie di valute preferite. Quando l'utente seleziona una coppia, viene visualizzato un nuovo ViewController per la conversione valutaria. L'utente può eliminare le preferenze toccando e trascinando le righe. Infine, la classe regola l'altezza delle righe della tabella.

***viewDidLoad:*** inizializza due array ***tableDataFromCurrency*** e ***tableDataToCurrency***, quindi recupera un dizionario di valute preferite da ***NSUserDefaults*** e lo assegna a ***countryCurrencies***. Successivamente, scorre *countryCurrencies* e aggiunge le valute di partenza e di destinazione ai rispettivi array. Infine, imposta i delegati *dataSource* e *delegate* della tabella per gestire la visualizzazione dei dati e gli eventi.

***tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section:*** Metodo richiamato per ottenere il numero di righe nella sezione della tabella.

***(UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath:*** Metodo per creare e restituire una cella per la tabella, impostandone il testo della cella con la valuta e il tasso di cambio corrispondenti.

***(void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath:*** Questo metodo viene chiamato quando l'utente seleziona una riga nella tabella. Crea un nuovo view controller ***ViewController***. Successivamente, istanzia un nuovo oggetto ***CurrencyConverter*** con le valute di partenza e di destinazione corrispondenti all'indice della riga selezionata nella tabella. Infine, presenta il nuovo view controller animato sulla schermata corrente.

***(BOOL)tableView:(UITableView *)tableView canEditRowAtIndexPath:(NSIndexPath *)indexPath:*** Metodo utilizzato per determinare se una riga specifica della tabella può essere modificata, restituisce sempre YES per consentire la modifica delle righe

***(CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath:*** Metodo utilizzato per specificare l'altezza di una riga specifica della tabella, restituendo sempre lo stesso valore per tutte le righe.

***tableView:(UITableView *)tableView commitEditingStyle:(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath *)indexPath:*** Questo metodo viene chiamato quando si compie una modifica alla riga della tabella, come l'eliminazione. Controlla se l'editingStyle è UITableViewCellEditingStyleDelete. Se sì, ottiene la coppia di valute (chiave e valore) relativa all'elemento da eliminare. Successivamente, individua l'indice dell'elemento nel dizionario countryCurrencies corrispondente alla coppia di valute. Rimuove quindi l'elemento dal dizionario countryCurrencies, salva il dizionario aggiornato nell'UserDefaults e rimuove l'elemento dalle strutture dati tableDataFromCurrency e tableDataToCurrency. Infine, ricarica la tabella per mostrare i cambiamenti.