**Weather App**

*Leonardo Babbucci*

*Matteo Cadoni*

*Sviluppo di applicazioni mobile*

****

Sommario

[Abstract 3](#_Toc134173291)

[Introduzione 3](#_Toc134173292)

[Modello 4](#_Toc134173293)

[Classe GPSCoordinates 4](#_Toc134173294)

[Classe Location 5](#_Toc134173295)

[Classe LocationDAO 6](#_Toc134173296)

[Classe LocationDB 7](#_Toc134173297)

[Classe LocationsHolder 8](#_Toc134173298)

[Classe TemperatureWorker 9](#_Toc134173299)

[Membri dati 9](#_Toc134173300)

[Costruttore 10](#_Toc134173301)

[Metodi 10](#_Toc134173302)

[Utilizzo della classe 10](#_Toc134173303)

[Classe Weather 10](#_Toc134173304)

[Classe WeatherType 11](#_Toc134173305)

[Classe OpeWeatherAPIService 11](#_Toc134173306)

[Metodi 11](#_Toc134173307)

[Utilizzo 11](#_Toc134173308)

[Classe ListFragment 12](#_Toc134173309)

[Classe DetailLocationFragment 12](#_Toc134173310)

[Classe DetailActivity 13](#_Toc134173311)

[Classe MainActivity 14](#_Toc134173312)

# Abstract

Il presente documento descrive il processo di sviluppo di un'applicazione mobile per il monitoraggio delle condizioni meteorologiche in diverse località. L'applicazione utilizza il GPS per rilevare la posizione corrente dell'utente e fornisce un elenco di località, incluse quelle aggiunte manualmente. Le informazioni sul tempo sono ottenute tramite API da OpenWeatherMap e vengono salvate in un database SQLite. L'applicazione controlla periodicamente le temperature e invia notifiche quando queste superano una determinata soglia.

# Introduzione

Il progetto di sviluppo di applicazioni mobile presentato in questo documento riguarda la creazione di un'applicazione per il monitoraggio delle condizioni meteorologiche in diverse località. L'obiettivo principale è di fornire all'utente un'interfaccia semplice e intuitiva per consultare le previsioni del tempo nelle località di interesse, utilizzando funzionalità come il GPS per rilevare la posizione corrente dell'utente e la possibilità di aggiungere manualmente nuove località.

L'applicazione è basata su un'architettura di tipo List-Detail, come illustrato nel mockup fornito. Per raccogliere i dati meteorologici, viene utilizzata l'API di OpenWeatherMap, una piattaforma che fornisce informazioni sulle condizioni del tempo in tempo reale. I dati ottenuti vengono poi salvati in un database SQLite, consentendo un rapido accesso alle informazioni e un controllo periodico delle temperature.

Uno dei requisiti fondamentali dell'applicazione è la capacità di inviare notifiche all'utente quando la temperatura in una località specifica scende o sale sopra una certa soglia, garantendo così che l'utente sia sempre aggiornato sulle condizioni meteorologiche. Per offrire un'esperienza utente ottimale, l'applicazione prevede anche il supporto per funzionalità aggiuntive come ViewPager e localizzazione.

La valutazione del progetto si basa su diversi fattori, tra cui la qualità del codice, il design dell'interfaccia utente e le funzionalità aggiuntive implementate. Al fine di soddisfare i requisiti di base, è fondamentale prestare attenzione alla gestione del GPS, del database SQLite e delle notifiche, elementi cruciali per garantire il corretto funzionamento dell'applicazione e l'efficacia delle sue funzioni principali.

In conclusione, il progetto proposto mira a sviluppare un'applicazione mobile efficiente e facile da usare per il monitoraggio delle condizioni meteorologiche in diverse località, sfruttando al meglio le potenzialità offerte dalle moderne tecnologie di sviluppo mobile.

# Modello

L'applicazione è strutturata attraverso l'utilizzo di diverse classi di modello, le quali svolgono un ruolo fondamentale nella gestione dei dati ottenuti da OpenWeatherMap e nella loro integrazione all'interno del sistema. Il loro scopo principale è quello di fornire una rappresentazione astratta e organizzata dei dati raccolti, facilitando così la loro manipolazione e utilizzo nell'ambito dell'applicazione.

Le classi di modello vengono utilizzate per modellizzare i dati scaricati da OpenWeatherMap tramite richieste HTTP, consentendo di estrarre e organizzare le informazioni rilevanti in modo strutturato e coerente. Queste classi fungono da interfaccia tra l'API di OpenWeatherMap e l'applicazione stessa, garantendo l'interoperabilità tra le diverse componenti del sistema.

Inoltre, le classi di modello svolgono un ruolo cruciale nella gestione del database, poiché permettono di creare, modificare e interrogare le tabelle contenenti i dati delle località e delle relative condizioni meteorologiche. Grazie a queste classi, l'applicazione è in grado di gestire efficacemente il flusso di dati tra il database e le componenti di visualizzazione e controllo, assicurando un'esperienza utente fluida e senza interruzioni.

Infine, le classi di modello sono responsabili della gestione dei dati utilizzati dall'applicazione nel corso delle sue operazioni. Ciò include la memorizzazione delle informazioni relative alle località, alle temperature e alle condizioni meteorologiche, nonché la gestione delle notifiche e dei processi di aggiornamento delle informazioni.

# Classe GPSCoordinates

La classe GPSCoordinates, presente nel package ch.supsi.dti.isin.meteoapp.model, è una classe che rappresenta una coppia di coordinate geografiche, ovvero latitudine e longitudine, utilizzate per identificare la posizione geografica di un punto sulla superficie terrestre. La classe è stata progettata per gestire e manipolare le coordinate GPS associate alle località per cui l'utente desidera visualizzare le informazioni meteorologiche.

La classe GPSCoordinates è composta dai seguenti attributi:

* mLatitude: un attributo di tipo double che rappresenta la latitudine del punto geografico.
* mLongitude: un attributo di tipo double che rappresenta la longitudine del punto geografico.

Il costruttore della classe GPSCoordinates accetta due parametri, latitude e longitude, che vengono utilizzati per inizializzare gli attributi mLatitude e mLongitude.

La classe GPSCoordinates fornisce i seguenti metodi pubblici per la gestione e l'accesso alle coordinate GPS:

* getLatitude(): restituisce il valore della latitudine associata all'oggetto GPSCoordinates.
* setLatitude(double latitude): imposta il valore della latitudine associata all'oggetto GPSCoordinates.
* getLongitude(): restituisce il valore della longitudine associata all'oggetto GPSCoordinates.
* setLongitude(double longitude): imposta il valore della longitudine associata all'oggetto GPSCoordinates.

Inoltre, la classe GPSCoordinates include un metodo toString() che restituisce una rappresentazione in formato stringa dell'oggetto GPSCoordinates, mostrando la latitudine e la longitudine con il seguente formato: "Latitude: <valore\_latitudine> Longitude: <valore\_longitudine>". Questo metodo è particolarmente utile per la visualizzazione delle coordinate in un formato leggibile e facilmente interpretabile.

In conclusione, la classe GPSCoordinates fornisce un'implementazione efficiente e flessibile per la gestione delle coordinate geografiche all'interno dell'applicazione. Essa permette di modellare e manipolare le informazioni relative alla posizione geografica delle località di interesse, garantendo un funzionamento efficace e preciso delle funzionalità legate alla geolocalizzazione e alle informazioni meteorologiche.

# Classe Location

La classe Location, situata nel package ch.supsi.dti.isin.meteoapp.model, è una classe che rappresenta una località geografica per la quale l'utente desidera ottenere informazioni meteorologiche. La classe è stata progettata per memorizzare e gestire le informazioni relative alle località di interesse, integrandosi con il database dell'applicazione tramite la libreria Android Room.

La classe Location presenta le seguenti annotazioni:

* @Entity(tableName = "location"): indica che la classe Location rappresenta una tabella del database chiamata "location".
* @PrimaryKey(): specifica che l'attributo Id è la chiave primaria della tabella.
* @ColumnInfo(name = "<nome\_colonna>"): assegna un nome alla colonna corrispondente all'attributo nella tabella del database.

Gli attributi della classe Location sono:

* Id: un attributo di tipo UUID che rappresenta l'identificativo univoco della località.
* mName: un attributo di tipo String che rappresenta il nome della località.
* mWeather: un attributo di tipo WeatherResponse, utilizzato per memorizzare le informazioni meteorologiche della località. L'annotazione @Ignore indica che questo attributo non verrà incluso nella tabella del database.

Il costruttore senza parametri della classe Location inizializza l'attributo Id generando un nuovo UUID casuale.

La classe Location fornisce i seguenti metodi pubblici per l'accesso e la gestione delle informazioni sulla località:

* getId(): restituisce l'UUID associato all'oggetto Location.
* setId(UUID id): imposta l'UUID associato all'oggetto Location.
* getName(): restituisce il nome della località associata all'oggetto Location.
* setName(String name): imposta il nome della località associata all'oggetto Location.
* getmWeather(): restituisce l'oggetto WeatherResponse contenente le informazioni meteorologiche della località.
* setmWeather(WeatherResponse mWeather): imposta l'oggetto WeatherResponse contenente le informazioni meteorologiche della località.

In sintesi, la classe Location è stata progettata per memorizzare e gestire le informazioni sulle località di interesse e le loro condizioni meteorologiche all'interno dell'applicazione. Essa si integra con il database tramite la libreria Android Room e offre un'implementazione efficiente e flessibile per il recupero e la manipolazione delle informazioni sulle località e le loro condizioni meteorologiche.

# Classe LocationDAO

La classe LocationDao, situata nel package ch.supsi.dti.isin.meteoapp.model, è un'interfaccia che definisce un Data Access Object (DAO) per la gestione delle operazioni sul database riguardanti gli oggetti della classe Location. L'interfaccia DAO offre una serie di metodi per eseguire operazioni CRUD (Create, Read, Update e Delete) sulle istanze della classe Location. Essa si integra con il database dell'applicazione tramite la libreria Android Room.

L'interfaccia LocationDao è annotata con @Dao, che indica a Room che si tratta di un'interfaccia DAO e consente l'implementazione automatica dei metodi definiti al suo interno.

L'interfaccia LocationDao definisce i seguenti metodi pubblici per la gestione delle località nel database:

* getLocations(): metodo che restituisce una lista di oggetti Location. L'annotazione @Query("SELECT \* FROM location") indica che il metodo eseguirà una query SQL per selezionare tutte le righe dalla tabella "location" e restituirle come oggetti Location.
* insertLocation(Location location): metodo che inserisce una nuova località nel database. L'annotazione @Insert indica che il metodo eseguirà un'operazione di inserimento per l'oggetto Location fornito come parametro.
* deleteLocation(Location location): metodo che elimina una località dal database. L'annotazione @Delete indica che il metodo eseguirà un'operazione di eliminazione per l'oggetto Location fornito come parametro.
* updateLocation(Location location): metodo che aggiorna una località esistente nel database. L'annotazione @Update indica che il metodo eseguirà un'operazione di aggiornamento per l'oggetto Location fornito come parametro.

In conclusione, l'interfaccia LocationDao fornisce un insieme di metodi per eseguire operazioni CRUD sulle località nel database dell'applicazione, utilizzando la libreria Android Room per gestire le interazioni con il database. Essa permette di manipolare e gestire le informazioni sulle località e le loro condizioni meteorologiche in modo efficiente e flessibile, semplificando l'integrazione tra l'applicazione e il database.

# Classe LocationDB

La classe LocationDB, situata nel package ch.supsi.dti.isin.meteoapp.model, è una classe astratta che estende RoomDatabase e rappresenta il database dell'applicazione per la gestione delle località. La classe LocationDB è stata progettata per creare, gestire e accedere al database dell'applicazione che memorizza le informazioni relative alle località e alle loro condizioni meteorologiche.

La classe LocationDB presenta l'annotazione @Database, che indica che si tratta di un database Room e specifica le entità incluse nel database (Location.class), la versione del database e se esportare o meno lo schema (exportSchema = false).

Gli attributi della classe LocationDB sono:

* instance: un'istanza statica di LocationDB, utilizzata per implementare il pattern Singleton e garantire che esista un'unica istanza del database.

La classe LocationDB definisce i seguenti metodi pubblici e astratti:

* locationDao(): un metodo astratto che restituisce un'istanza dell'interfaccia LocationDao, utilizzata per eseguire operazioni CRUD sulle località nel database.

Inoltre, la classe LocationDB include un metodo statico e sincronizzato getInstance(Context context), che implementa il pattern Singleton. Il metodo verifica se esiste già un'istanza del database; in caso contrario, ne crea una nuova utilizzando il metodo databaseBuilder() di Room, specificando il contesto dell'applicazione, la classe del database e il nome del file del database ("location\_db"). Il metodo fallbackToDestructiveMigration() consente di eseguire una migrazione distruttiva in caso di aggiornamenti del database non supportati.

In sintesi, la classe LocationDB fornisce un'implementazione efficace e flessibile per la creazione, la gestione e l'accesso al database delle località dell'applicazione. Essa utilizza la libreria Android Room per integrarsi con il database e offre un'interfaccia semplice e intuitiva per interagire con le informazioni sulle località e le loro condizioni meteorologiche.

# Classe LocationsHolder

La classe LocationsHolder, situata nel package ch.supsi.dti.isin.meteoapp.model, è un singleton che gestisce l'accesso alle località e alle loro informazioni meteorologiche. È responsabile della gestione delle operazioni relative alle località, come l'aggiunta, la rimozione e la ricerca delle località.

Gli attributi della classe LocationsHolder sono:

* sLocationsHolder: un'istanza statica di LocationsHolder, utilizzata per implementare il pattern Singleton.
* db: un'istanza del database LocationDB, utilizzata per accedere al database delle località.
* mExecutor: un Executor per eseguire operazioni in un thread separato.
* mLocations: una lista di oggetti Location che rappresenta le località gestite dall'applicazione.
* mCallbacks: una lista di interfacce LocationsLoadedCallback, utilizzate per notificare agli oggetti interessati quando le località sono state caricate.

La classe LocationsHolder definisce l'interfaccia LocationsLoadedCallback, che contiene un singolo metodo onLocationsLoaded(List<Location> locations) utilizzato per notificare quando le località sono state caricate.

Il costruttore privato LocationsHolder(Context context) inizializza l'istanza del database, l'Executor e le liste mLocations e mCallbacks. Chiama inoltre il metodo fetchLocationsFromDatabase() per recuperare le località dal database.

I metodi pubblici della classe LocationsHolder includono:

* get(Context context): un metodo statico che restituisce l'istanza di LocationsHolder, creandola se non esiste.
* addLocationsLoadedCallback(LocationsLoadedCallback callback): aggiunge un oggetto callback alla lista mCallbacks, o chiama immediatamente onLocationsLoaded(mLocations) se mLocations non è vuota.
* getCurrentLocation(Context context): restituisce le coordinate GPS dell'utente utilizzando la libreria SmartLocation.
* addLocationDB(Location location): aggiunge una località al database e alla lista mLocations.
* removeLocation(Location location): rimuove una località dal database e dalla lista mLocations.
* getLocations(): restituisce la lista delle località mLocations.
* getLocation(UUID id): restituisce un oggetto Location dalla lista mLocations in base al suo UUID, oppure null se non viene trovato.

La classe LocationsHolder contiene anche metodi privati come fetchLocationsFromDatabase() per recuperare le località dal database e aggiungerle alla lista mLocations, e getLocalLocation(Context context, final Location location) per ottenere le coordinate GPS di una località specifica.

In sintesi, la classe LocationsHolder è un componente chiave dell'applicazione MeteoApp, che gestisce l'accesso e la manipolazione delle località e delle loro informazioni meteorologiche. Utilizza la libreria Android Room per accedere al database delle località e fornisce un'interfaccia semplice e intuitiva per interagire con le località e le loro informazioni.

# Classe TemperatureWorker

Ciao La classe TemperatureWorker estende la classe Worker di Android e si occupa di monitorare i cambiamenti di temperatura e inviare notifiche all'utente in caso di variazioni significative.

## Membri dati

* **private Context mContext**: Un oggetto **Context** che rappresenta il contesto in cui il worker viene eseguito.
* **private static final String TAG**: Un tag utilizzato per il logging.
* **private LocalDateTime lastNotificationTime**: L'ultima volta che è stata inviata una notifica.

## Costruttore

Il costruttore **TemperatureWorker(@NonNull Context context, @NonNull WorkerParameters params)** accetta due parametri: il contesto in cui il worker viene eseguito e i parametri del worker. Inizializza il contesto e imposta l'ultimo orario di notifica a un minuto fa.

## Metodi

* **@NonNull @Override public Result doWork()**: Questo metodo viene chiamato quando il worker viene eseguito. Recupera le coordinate geografiche correnti utilizzando la classe **LocationsHolder**, controlla la temperatura e invia una notifica se necessario. In caso di successo, restituisce **Result.success()**, altrimenti **Result.failure()**.
* **private void checkTemperature(GPSCoordinates coordinates) throws IOException**: Questo metodo privato controlla la temperatura utilizzando le coordinate geografiche fornite e invia una notifica se la temperatura è cambiata significativamente. Utilizza la libreria Retrofit per effettuare una chiamata all'API OpenWeatherMap e ottiene la temperatura corrente.
* **private void sendNotification(double currentTemperature, double previousTemperature)**: Questo metodo privato invia una notifica all'utente riguardo ai cambiamenti di temperatura. Accetta due parametri, la temperatura corrente e la temperatura precedente, e crea il testo della notifica basato sulla differenza tra le due temperature. Successivamente, costruisce e invia la notifica utilizzando **NotificationCompat.Builder** e **NotificationManagerCompat**.

## Utilizzo della classe

La classe **TemperatureWorker** viene utilizzata per monitorare i cambiamenti di temperatura e inviare notifiche all'utente in caso di variazioni significative. Essa viene eseguita come un worker nel contesto dell'applicazione e utilizza le coordinate geografiche fornite dalla classe **LocationsHolder** per ottenere la temperatura corrente tramite l'API OpenWeatherMap.

# Classe Weather

Il codice fornito è una classe Java chiamata `Weather` che definisce le proprietà e i metodi relativi ai dati meteorologici.

La classe contiene i dati meteorologici, come temperatura, pressione, umidità, velocità del vento, nuvole e volume delle precipitazioni/nevicate. A queste variabili si accede tramite metodi pubblici getter e setter.

La classe include anche un tipo enum `WeatherType` e una variabile di istanza `mWeatherType` di questo tipo, che rappresenta le condizioni meteorologiche generali, come sole, nuvoloso, ecc.

# Classe WeatherType

Questo enum è stato implementato per rendere dinamico il caricamento delle immagini in base allo stato sul meteo restituito dalle API. Contiene la descrizione e la relativa icona. Viene utilizzato come se fosse una Map chiave valore.

# Classe OpeWeatherAPIService

L'interfaccia **OpenWeatherAPIService** definisce i metodi per effettuare richieste alle API di OpenWeatherMap per ottenere le informazioni sulle condizioni meteorologiche.

## Metodi

* **@Headers({"Accept: application/json", "Content-Type: application/json"})** **@GET("weather")** **Call<WeatherResponse> getWeatherByCity(@Query("q") String city, @Query("appid") String api\_key);**

Questo metodo esegue una richiesta GET all'endpoint "weather" delle API di OpenWeatherMap per ottenere le informazioni sulle condizioni meteorologiche di una città specificata in input tramite il parametro **city**. Il parametro **api\_key** rappresenta la chiave API necessaria per effettuare le richieste. Ritorna un oggetto **Call<WeatherResponse>** che contiene la risposta del server, che può essere convertita in un oggetto **WeatherResponse** contenente i dati delle condizioni meteorologiche.

* **@Headers({"Accept: application/json", "Content-Type: application/json"})** **@GET("weather")** **Call<WeatherResponse> getWeatherByCoordinates(@Query("lat") double lat, @Query("lon") double lon, @Query("appid") String api\_key);**

Questo metodo esegue una richiesta GET all'endpoint "weather" delle API di OpenWeatherMap per ottenere le informazioni sulle condizioni meteorologiche utilizzando le coordinate geografiche fornite tramite i parametri **lat** e **lon**. Il parametro **api\_key** rappresenta la chiave API necessaria per effettuare le richieste. Ritorna un oggetto **Call<WeatherResponse>** che contiene la risposta del server, che può essere convertita in un oggetto **WeatherResponse** contenente i dati delle condizioni meteorologiche.

## Utilizzo

L'interfaccia **OpenWeatherAPIService** viene utilizzata in combinazione con la libreria Retrofit per semplificare il processo di effettuare richieste HTTP e deserializzare automaticamente la risposta JSON in oggetti Java. Per utilizzare questa interfaccia, è necessario creare un'istanza di **Retrofit** configurata con l'URL di base delle API di OpenWeatherMap e un convertitore Gson. Dopodiché, è possibile creare un'istanza dell'interfaccia **OpenWeatherAPIService** chiamando il metodo **create()** dell'istanza di **Retrofit**.

# Classe ListFragment

La classe **ListFragment** è un componente dell'applicazione MeteoApp che estende la classe **Fragment**. Il suo scopo principale è di visualizzare una lista di località e le loro informazioni meteorologiche in tempo reale. Per farlo, utilizza un **RecyclerView** che viene popolato con oggetti **Location** recuperati dal **LocationsHolder**.

Quando il fragment viene creato, il metodo **onCreateView** viene invocato, e la vista del fragment viene inizializzata. Il **RecyclerView** viene quindi impostato con un **LinearLayoutManager** e il **LocationsHolder** viene configurato per caricare le località e aggiornare l'interfaccia utente una volta completato il caricamento.

Il metodo **updateUI** viene utilizzato per aggiornare l'interfaccia utente del fragment. Se l'adapter del **RecyclerView** non è stato ancora creato, viene istanziato un nuovo **LocationAdapter** e impostato come adapter del **RecyclerView**. In caso contrario, l'adapter viene aggiornato con le nuove località e notificato del cambiamento.

Il fragment contiene anche un menu che permette all'utente di aggiungere una nuova località attraverso un **AlertDialog**. Quando l'utente inserisce il nome di una città e conferma, viene invocata la funzione **getWeatherByCityName** che effettua una chiamata alle API di OpenWeatherMap per ottenere le informazioni meteorologiche della città specificata. Se la chiamata ha successo, viene aggiornata l'interfaccia utente e il nuovo oggetto **Location** viene aggiunto al **LocationsHolder**. In caso di errore, viene mostrato un messaggio di errore all'utente.

La classe **LocationHolder** estende **RecyclerView.ViewHolder** e gestisce il layout e il comportamento di un singolo elemento nella lista delle località. La classe **LocationAdapter** è un adapter personalizzato che estende **RecyclerView.Adapter<LocationHolder>**, e gestisce il processo di creazione e aggiornamento degli oggetti **LocationHolder** all'interno del **RecyclerView**.

# Classe DetailLocationFragment

La classe **DetailLocationFragment** è un frammento di Android che mostra le informazioni sulle condizioni meteorologiche per una determinata posizione. La posizione può essere specificata tramite coordinate GPS (latitudine e longitudine) o tramite nome della città. Utilizza l'API OpenWeatherMap per ottenere le informazioni meteorologiche.

Le principali parti del codice sono:

1. Le costanti e le variabili di istanza, tra cui i riferimenti alle viste (come **cityName**, **temperature**, **description** e **weatherIcon**), il modello di posizione **mLocation** e i Runnable utilizzati per aggiornare l'interfaccia utente.
2. I metodi **newInstance()**, che creano un'istanza di **DetailLocationFragment** con argomenti specificati come coordinate GPS o nome della città.
3. Il metodo **onCreate()**, che inizializza il modello di posizione e richiama i metodi per ottenere informazioni sulla posizione e sulle condizioni meteorologiche.
4. I metodi **getLocationFromGPS()** e **getLocationFromName()**, che utilizzano Retrofit per effettuare chiamate asincrone all'API OpenWeatherMap e ottenere informazioni sulle condizioni meteorologiche in base alle coordinate GPS o al nome della città, rispettivamente.
5. Il metodo **updateUI()**, che aggiorna l'interfaccia utente con le informazioni meteorologiche ricevute, come il nome della città, la temperatura, la descrizione del tempo e l'icona del tempo.
6. Il metodo **kelvinToCelsius()**, che converte la temperatura da Kelvin a Celsius.
7. Il metodo **onCreateView()**, che crea la vista del frammento e inizializza i riferimenti alle viste.

Nel complesso, questa classe crea un frammento Android che mostra le informazioni sulle condizioni meteorologiche per una determinata posizione, utilizzando le coordinate GPS o il nome della città come input e l'API OpenWeatherMap per ottenere i dati sulle condizioni meteorologiche.

# Classe DetailActivity

La classe **DetailActivity** è un'attività Android che ospita il frammento **DetailLocationFragment** per mostrare le informazioni sulle condizioni meteorologiche di una determinata posizione. La posizione può essere specificata tramite un ID, coordinate GPS (latitudine e longitudine) o tramite il nome della città.

Le principali parti del codice sono:

1. Le costanti **EXTRA\_LOCATION\_ID**, **EXTRA\_GPS\_LATITUDE** e **EXTRA\_GPS\_LONGITUDE**, che sono usate per passare i dati tramite l'intento che avvia l'attività.
2. I metodi **newIntent()**, che creano un intent per avviare **DetailActivity** con argomenti specificati come ID della posizione o coordinate GPS.
3. Il metodo **onCreate()**, che si occupa della logica di avvio dell'attività e dell'inizializzazione del frammento **DetailLocationFragment**:
   * Controlla se l'intento ha un extra **EXTRA\_LOCATION\_ID**. In caso affermativo, cerca la posizione utilizzando l'ID e crea un'istanza di **DetailLocationFragment** utilizzando la latitudine e la longitudine o il nome della città, a seconda della disponibilità dei dati.
   * In caso contrario, controlla se l'intento ha gli extra **EXTRA\_GPS\_LATITUDE** e **EXTRA\_GPS\_LONGITUDE**. Se presenti, crea un'istanza di **DetailLocationFragment** utilizzando le coordinate GPS.
   * Infine, se il frammento non è già presente nella gerarchia, aggiunge il nuovo frammento **DetailLocationFragment** al **FragmentManager**.

Nel complesso, questa classe crea un'attività Android che ospita il frammento **DetailLocationFragment** per visualizzare le informazioni sulle condizioni meteorologiche di una determinata posizione, utilizzando l'ID della posizione, le coordinate GPS o il nome della città come input.

# Classe MainActivity

La classe **MainActivity** è un'attività Android che contiene principalmente un frammento **ListFragment** per visualizzare un elenco di posizioni e le loro informazioni meteorologiche, e un pulsante per avviare un'attività **DetailActivity** basata sulla posizione geolocalizzata dell'utente.

Le principali parti del codice sono:

1. La variabile **coordinates** che memorizza le coordinate GPS dell'utente e la variabile **recentTemperature** che memorizza la temperatura recente.
2. Il metodo **onCreate()** che si occupa della logica di avvio dell'attività:
   * Richiede i permessi necessari per accedere alla posizione dell'utente e alla connessione Internet.
   * Crea un'istanza del database **LocationDB**.
   * Crea un canale di notifica se l'app viene eseguita su Android Oreo o versioni successive.
   * Configura e avvia un **PeriodicWorkRequest** per aggiornare la temperatura ogni minuto utilizzando la classe **TemperatureWorker**.
3. Il metodo **requestPermissions()** che richiede i permessi necessari all'applicazione.
4. Il metodo **onRequestPermissionsResult()** che gestisce la risposta dell'utente alla richiesta di permessi e avvia l'attività se i permessi sono stati concessi.
5. Il metodo **open()** che viene chiamato una volta ottenuti i permessi:
   * Imposta il layout dell'attività.
   * Avvia la geolocalizzazione dell'utente.
   * Configura il pulsante "geolocateButton" per avviare l'attività **DetailActivity** utilizzando le coordinate GPS dell'utente.
   * Aggiunge il frammento **ListFragment** al **FragmentManager** se non è già presente.
6. Il metodo **monitorWorkStatus()** che monitora lo stato del lavoro **TemperatureWorker**.
7. Il metodo **geolocate()** che ottiene le coordinate GPS dell'utente utilizzando la classe **LocationsHolder**.

In sintesi, questa classe crea un'attività Android che mostra un elenco di posizioni e le loro informazioni meteorologiche tramite un **ListFragment**, e permette all'utente di visualizzare le informazioni meteorologiche sulla sua posizione corrente tramite un pulsante che avvia un'attività **DetailActivity**. L'attività richiede anche i permessi necessari e avvia un lavoro periodico per aggiornare la temperatura.