Università degli Studi di Napoli Federico II

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica LM

Corso di Ingegneria del Software II

A.A 2013/2014

Prof. Porfirio Tramontana

Documentazione dell'Applicazione Android *"Luna Rossa"*

Sajmir Rusi M63/42

s.rusi@strudenti.unina.it

Sommario

Introduzione 3
L'Idea 3
Sviluppo 4
L'applicazione 4
Fenomeno della Luna Rossa 4
Delta-T 5
Tecnologie e Strumenti utilizzati 6
Panoramica 6
Activities 7
- RedMoo 7
- Terra 7
- Sole 7
- La classe Luna 7
- PlacePicker 7

Classes 8
- WeatherHttpClient 8
- DataCircumstances 8
- DataEntry 8
- LocalEventData 8
- JSONWeatherParser 8
- LocationRetrive 8
- Weather 8

Connessione e ricezione dati da OpenWeather 8
Uso di Google Autocomplete e ricezione suggerimenti da Google Places 12
Calcolo del eclisse 13
Visualizzazione del evento 14
Activity Sole 15
Bibliografia 16

Introduzione

L'Idea

Oggetto dell'elaborato è lo sviluppo di un'Applicazione Android per il calcolo del verificarsi della **Luna Rossa** nella **propria città**.

L'applicazione oltre a fornire l'indicazione sull'orario d'inizio e fine eclisse nel caso di presenza di un evento luna rossa nel giorno presente, offre anche un servizio Meteo. Il servizio meteo si basa sulle API fornite da OpenWeather.org .

Le condizioni metereologiche comprendono:

- Temperatura minima media e massima
- Nuvolosità o Copertura
- Pressione atmosferica
- Umidità
- Velocità del vento

Inoltre è disponibile anche l'orario di sorgere e tramonto del sole.

L'utente ha la possibilità di inserire una località a piacere, per la quale ricevere le indicazioni meteo, il verificarsi della luna rossa e l'orario di sorgere e tramonto del solo.

La scelta della località è stata agevolata con l'integrazione di Google Places che suggerisce una lista di località mentre digitiamo.

Tutte le funzionalità si basano sulla posizione geografica dove si trova il dispositivo, questa informazioni viene recuperata dal servizio GPS del dispositivo e ove esso non sia disponibile o disabilitato, viene preso l'ultima location valida nota, fino a una successivo aggiornamento della stessa.

Figura 1 Activity principale "Terra"

Sviluppo

La natura dell'applicazione ha richiesto la suddivisione dello sviluppo in diverse macro fasi

- Studio delle metodologie di sviluppo Android
- Studio del implementazione delle Google Api (place finder e autocomplete)
- Studio del fenomeno della luna rossa e dei metodi di calcolo del deltaT
- Studio delle api di OpenWeather.org
- Implementazione del calcolatore della luna rossa in Java
- Testing del applicazione calcolatore
- Implementazione della applicazione Android
- Integrazione del calcolatore all'interno dell'app Android

L'applicazione

Fenomeno della Luna Rossa

L'applicazione è nata con l'idea principale di segnalare all'utente il giorno in cui si può vedere il fenomeno della luna rossa dalla propria città.

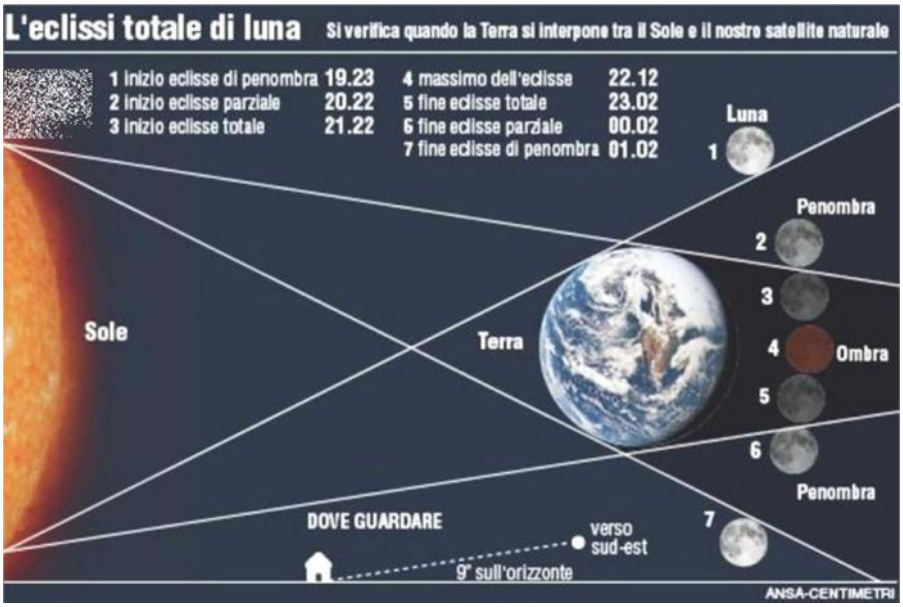

Figura 2 Eclissi lunare, posizione relativa del sole e della luna

(wiki)La Luna rossa, in astronomia, è un fenomeno ottico di rifrazione e di scattering di Rayleigh che si attua durante le eclissi di Luna.

La luce solare attraversa l'atmosfera terrestre e subisce una rifrazione differenziale (dal viola cupo al rosso scuro). I raggi sono multicolori perché l'atmosfera terrestre si comporta come un prisma. Tali raggi in parte

colpiscono la superficie del nostro pianeta il quale li rinvia sulla Luna. I colori rossi sono quelli meno diffusi dall'aria, al contrario dei blu-violetti, che sono invece "sparpagliati" in tutte le direzioni, ed è questo il motivo per cui il cielo è azzurro e il sole all'orizzonte appare rosso.

La Luna, che durante l'eclisse non è raggiunta dalla luce diretta del Sole, è pertanto illuminata da questa luce rifratta dall'atmosfera terrestre, in prevalenza rossa, e quindi appare rossa.

La luna bassa sull'orizzonte appare rossa per lo scattering di Rayleigh

Tutto ciò spiega (in parte) il fenomeno dell'arrossamento lunare. A complicarlo c'è la colorazione della superficie terrestre sulla quale insistono i raggi che sono riflessi sulla Luna: gli oceani, i deserti, le foreste e così via, si presentano con colorazioni diverse le quali, combinandosi con il rosso possono dare luogo ad altrimenti impreviste tonalità color rame, marrone, ecc. e per di più mutevoli nel trascorrere del tempo.

Nel linguaggio comune, invece, la “luna rossa” è quella che si vede grande e bassa sull'orizzonte. Rossa per lo stesso motivo per cui il sole all'orizzonte è rosso, e grande per un'illusione ottica. Di nuovo, la stessa illusione ottica che ci fa apparire grande il sole all'orizzonte (mentre la sua forma schiacciata è dovuta alla rifrazione).

Delta-T

(NASA, deltaT)Per molti secoli, l'unità fondamentale del tempo è stato la rotazione periodica della Terra rispetto al sole. Il Tempo Universale o UT (conosciuto come Greenwich Mean Time o GMT) si basa sul tempo solare medio di Greenwich, in Inghilterra. Purtroppo, l’UT non è una scala di tempo uniforme, perché il periodo di rotazione della Terra è in graduale diminuzione.

Il Terrestrial Dynamical Time (TDT) invece è una scala di tempo atomico, può essere pensato come il tempo che sarebbe stato tenuto da un orologio ideale. La maggior parte dei calcoli astronomici (comprese le eclissi) utilizzano TDT poiché le orbite di tutti i pianeti possono essere descritti accuratamente con esso.

Anche se le previsioni eclissi solari(e quelli lunari) sono basate sul TDT, la posizione del percorso dell'eclisse dipende ancora dal UT. Per convertire le previsioni fatte con TDT in previsioni osservabili da UT, bisogna conoscere la differenza tra TDT e UT.

Questo importante parametro in astronomia è noto come delta-T o ΔT (ΔT = TDT - UT). Il ΔT è stato utilizzato per ottenere delle previsioni molto accurate dei eclissi lunari.

Tecnologie e Strumenti utilizzati

Per lo sviluppo dell'itera applicazione *Android* sono stati utilizzati i seguenti strumenti:

- Android SDK: kit di sviluppo che fornisce le librerie API e gli strumenti necessari a costruire, testare e fare il debug di un app Android
  - Eclipse + Android Developer Tools ADT plugin: piattaforma di sviluppo per la creazione di applicazioni android
  - Strumenti di supporto di Android SDK
  - Emulatore per Android
- Libreria google-play-services: consente di utilizzare tutte le API di Google
- Org.json e json-simple per la manipolazione di oggetti json
- Le API di Openweather.org, consentono la ricezione di dati metereologiche aggiornate in base alla città.
- Google Autocomplete per la ricerca di assistita di luoghi
- Google Place Finder: conente di trovare una location

Panoramica

L'applicazione è strutturata in modo tale da mettere a disposizione le principali funzionalità di visualizzazione delle previsioni meteo e il verificarsi della luna rossa, in accordo con le caratteristiche del sistema operativo Android il quale prevede lo sviluppo di due entità trincipali, le *Activity* e i *Service*. Le prime costituiscono le interfacce di interazione con l'utente mentre i Service sono dei componenti per l'esecuzione di operazioni di background.

Il ciclo di vita delle Activity è stato modellato sulla base delle operazioni che un utente deve poter effettuare

interagendo con l'applicazione, in virtù della funzionalità e della fruibilità della stessa. In Figura 3 Architettura del Applicazione Luna Rossaè mostrata l'architettura dell'applicazione.

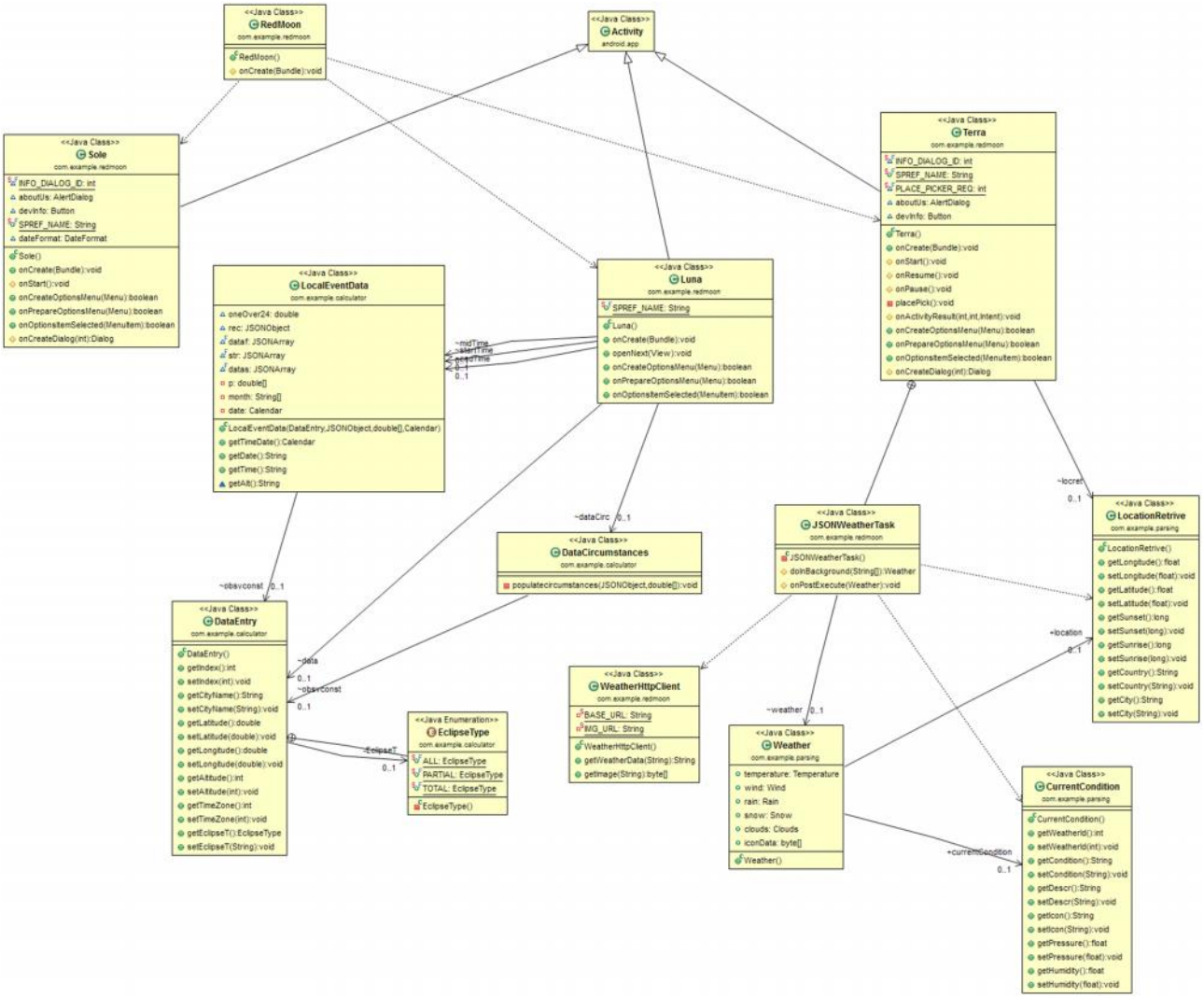

Figura 3 Architettura del Applicazione Luna Rossa

Activities

- **RedMoo***:* Utilizzata per realizzare Estende la classe TabActivity per realizzare i tre "Tab" di selezione delle funzionalità del applicazione
- **Terra** è l'Activity di avvio dell'applicazione. Ha la responsabilità di controllare prima di tutto la presenza di una connessione dati e in caso negativo di avvisare l'utente di abilitarne una.
- **Sole** è l'Avtivity che si occupa della visualizzazione del orario di tramonto e sorgere del sole.
- **La classe Luna** estende la classe Activity, richiede l'elaborazione dei dati alla classe LocalEventData
- **PlacePicker** Implementa la funzionalità di suggerimento della ricerca di un posto

Classes

- **WeatherHttpClient *Class*** crea la connessione con il servizio OpenWeather.org e ritorna i dati ottenuti
- **DataCircumstances** Elabora i dati provenienti dal sistema android e quelli ottenuti dal JSON di riferimento, conservando il tutto in una struttura ad array.
- **DataEntry** la classe crea un oggetto contenente i dati di geo localizzazione.
- **LocalEventData** classe responsabile del calcolo del giorno, del ora di inizio e fine di un evento di eclisse
- **JSONWeatherParser** Riceve l'oggetto JSON, fa il parsing e Inizializza i diversi oggetti della classe Weather
- **LocationRetrive** Oggetto
- **Weather** Crea e inizializza i diversi oggetti: Temperature, Condition e Wind

Connessione e ricezione dati da OpenWeather

Alla creazione del Activity Terra, dopo la verifica della presenza di una connessione dati valida, viene sfruttato il Location Manager per recuperare delle coordinata geografica o il metodo getCity() della classe LocationRetrive per recuperare l'ultimo nome della città precedentemente salvato.

Nel caso in cui non c'è una connessione dati attiva viene visualizzato un messaggio di errore e viene invitato l'utente ad attivarne una, come si può vedere in Figura 4 Connesione dati mancante

Ottenute queste informazioni viene lanciato in esecuzione un JSONWeather Task che crea la query string

```
task.execute(new String[]{"lat=" + String.valueOf(gps[0]), "lon=" + String.valueOf(gps[1]), city });
```

la quale verrà utilizzata per la richiesta dei dati meteo.

L'invio della richiesta viene effettuato da un oggetto della classe WeatherHttpClient cha appunto si prende carico dell'invio della richiesta e ricezione dei dati.

```
if(params[2] != null)
        {data = ( (new WeatherHttpClient()).getWeatherData(url_extension_city));}
else{data = ( (new WeatherHttpClient()).getWeatherData(url_extension));}
```

La classe **WeatherHttpClient** usa **`HttpURLConnection`** `per preparare la URL, instanziare una connessione http, inviare una chiamata http Get e riceve la risposta.`

Ottenuto la risposta viene ritornato un oggetto String al JSONWeather Task che precedentemente ha fatto la richiesta e la connessione viene chiusa.

Come da specifiche delle API OpenWeather (Weather, 2014) la richiesta delle condizioni metereologiche può essere fatta in diversi modi, quella utilizzata in questo progetto prevede un indirizzo base detto all'end poin riportato qui di seguito e una query string che contiene i parametri della richiesta.

```
private static String BASE_URL = "http://api.openweathermap.org/data/2.5/weather?";
```

La query string viene composta in due modi differenti a seconda della disponibilità delle coordinate gps presenti nel dispositivo e della location ottenuta da Google Places:

Nel caso in cui si ha l'informazione di nome della città al parametro q viene assegnato tale valore

- `String url_extension_city = "q=" + params[2] + "&lang=it";`

Se invece si conoscono le coordinate geografiche del luogo la query string viene composta in quest'altro modo.

- `String url_extension = params[0] + "&" + params[1] + "&lang=it";`

Il Task chiamato per gestire la ricezione dei dati, ottenuta l'oggetto String, chiama il medoto getWeather della classe JSONWeatherParser che si occupa di parsare la stringa, creando dei oggetti della classe Weather e inizializzando opportunamente gli oggetti:

```
public CurrentCondition currentCondition = new CurrentCondition();
public Temperature temperature = new Temperature();
public Wind wind = new Wind();
public Rain rain = new Rain();
public Snow snow = new Snow() ;
public Clouds clouds = new Clouds();
```

questi oggetti vengono usati per creare l'UI del Activity.

La struttura delle Classi coinvolte in questa Activity viene riportata in figura

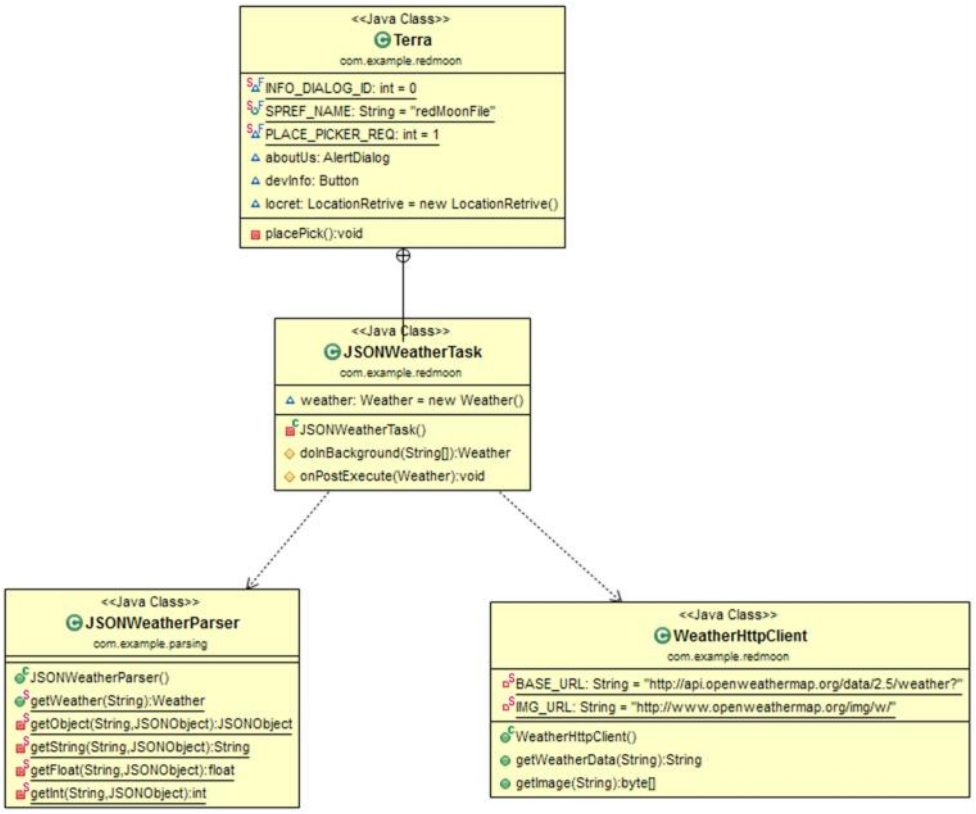

Figura 5 Strura delle classi coinvolte in Activity "Terra"

La Classe **Terra** è l'Activity che viene visualizzata all'avvio del applicazione, le informazioni sul meteo vengono visualizzate subito dopo aver ricevuto la risposta da OpenWeather.org e ovviamente dopo che la risposta è stata parsata.

```
<activity
    android:name="com.example.redmoon.RedMoon"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

In questa Activity è stato creato anche un istanza di **SharedPreferences,** un metodo semplice per condividere delle informazioni all'interno dell'applicazione. L'intento è di poter condividere le informazioni come il nome della città o le coordinate geografiche.

È stato pensato di inserire la creazione e l'inizializzazione di questo oggetto in una parte del lifecycle dalla applicazione dove non influisce sulle prestazione della stessa.

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

```
            SharedPreferences settings = getSharedPreferences(SPREF_NAME, 0);
            SharedPreferences.Editor editor = settings.edit();
            editor.putString("lastCity", locret.getCity() + ", "
+locret.getCountry());
            editor.putLong("lastsunrise", locret.getSunrise());
            editor.putLong("lastsunset", locret.getSunset());
            editor.putFloat("lat", locret.getLatitude());
            editor.putFloat("lon", locret.getLongitude());
            // Commit the edits!
            editor.commit();
      }
```

Uso di Google Autocomplete e ricezione suggerimenti da Google Places

A partire dal *Activity* Terra è possibile accede anche alle funzionalità delle **Google Places e Google Autocomplate**; Ovvero alla possibilità di trovare una location in modo assistito da Google places.

Figura 6 Google Autocomlete & Google Places

Al click sul nome della città visualizzato nel header del **Activity** viene portato in cima allo stack **l'Activity PlacePicker** che implementa l'interfaccia **OnItemClickListener**

```
cityText.setOnClickListener(new OnClickListener(){
      @Override
      public void onClick(View v) {
      placePick();
      cityText = (TextView) findViewById(R.id.cityText);
      }
});
```

Questa **Activity** permette la visualizzazione di una lista di location disponibili per la selezione, tale lista viene aggiornata in un certo intervallo modificabile, dopo aver inserito un nuovo carattere.

Individuata la location desiderata, la **TextView** e le condizioni metereologiche vengono aggiornate.

L'informazione riguardo alla location viene salvata nelle **`SharedPreferences`** per poter essere utilizzate dalle altre **Activities**.

Calcolo del eclisse

La parte puramente di calcolo dei eventi è contenuta all'interno del package *com.example.calculator*

Il calcolo dei eventi di eclissi totale della luna è basato su dati provenienti da una fonte ufficiale di Nasa. (NASA, JLEX-EU)

Sono stati recuperati, manipolati e convertiti in oggetto JSON i dati che riguardano il centenario 2001-2100. L'elemento basilare di questi dati è il deltaT che è stato presentato nel paragrafo Delta-T

La classe LocalEventData ottenute le informazioni sulla locatio calcola la data e l'orario di un evento di eclisse totale della luna.

La struttura delle classi coinvolte nel calcolo del eclisse è presentata in Figura 7 Struttura delle classi coinvolte nel Activity "Luna"

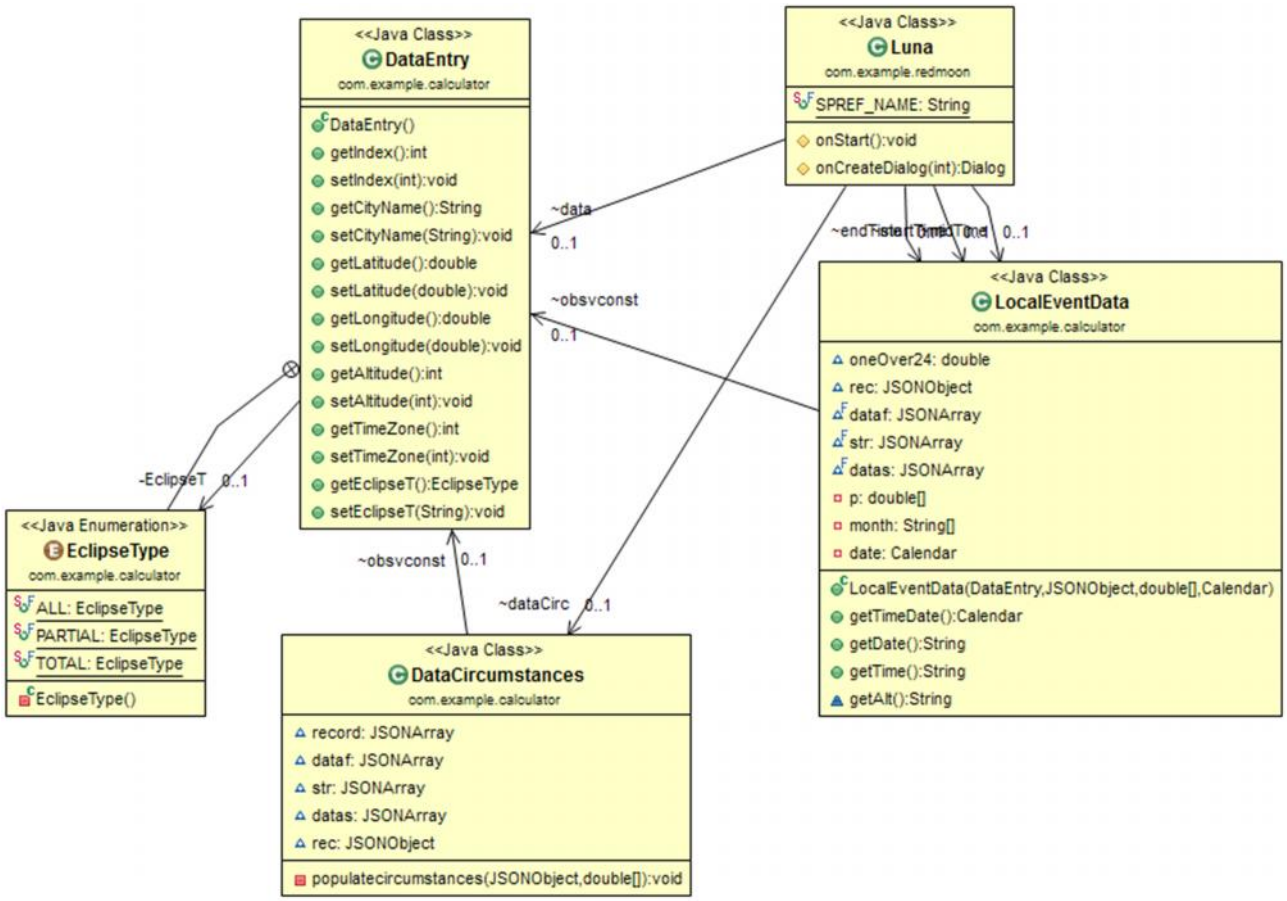

Figura 7 Struttura delle classi coinvolte nel Activity "Luna"

Vengono recuperate le informazioni di localizzazione del dispositivo attraverso un oggetto della classe DataEntry e le informazioni sui eventi di eclisse globali compresi di deltaT.

```
public LocalEventData(DataEntry data,  JSONObject rec, double [] p, Calendar date) {

		this.rec = rec;
		this.obsvconst = data;
		this.p = p;
		this.dataf = (JSONArray) rec.get("data");
		this.str = (JSONArray) rec.get("str");
		this.datas = (JSONArray) rec.get("datas");
		this.date = date;
	}
```

I due metodi principali dell'applicazione sono **getTime()** e **getDate()**

Il primo metodo calcola l'ora di inizio eclisse, l'ora di massimo eclisse visibile e l'ora di fine eclisse.

La data di un evento invece viene calcolato da getData() e restituito sotto forma di stringa.

In questo package troviamo anche la classe Main.java che è stata utilizzata per lo sviluppo e il testing del applicazione Java, prima di essere integrato nella app redMoon.

Visualizzazione del evento

Nella Activity Luna viene istanziato un oggetto LocalEventData e tramite i metodi getTime() e getDate() della stessa classe è possibile recuperare l'evento del giorno nel caso in cui ci sia uno:

```
	if(day == evDay && month == evMonth && year == evYear){
	//Se la condizione è vera, abbiamo un evento
		startTimeV.setText(startH);
		midTimeV.setText(midH);
		endTimeV.setText(endH);

		i =+ record.size();
	}else{
		noEclipseV.setText(R.string.noEclipse);
```

nella presente implementazione viene eseguito un ciclo

```
	for(int i = 0; i < record.size();i++)
```

su tutto l'array JSON e questo impatta sulle prestazioni dell'applicazione. In una successiva release si implementerà un differente algoritmo di ricerca.

Nel caso di presenza di un evento nel giorno presente viene visualizzata l'interfaccia che visualizza l'ora di inizio di massimo e di fine, altrimenti viene visualizzato un testo che informa sull'assenza di evento.

Figura 8 A-Nel caso di presenza di un evento **B-Se non c'è un evento** **C-Alla pressione sul pulsante**

Il pulsante *"Next RedMoon"* apre una finesta di Dialog con la data della prossima Luna Rossa se nel anno in corso sarà visibile una dalla propria città e in caso contrario presenta un messaggio che informa su questa eventualità.

Activity Sole

Nella presenta release questa activity presenta una semplice funzionalità: l'ora precisa quando il sole sorge e quando tramonta. Il dato viene precedentemente inizializzato dalla Activity "Terra" è ricevuto dal servizio di Open Weather. Questa funzionalità a prima vista insignificante a prima vista può essere molto importante per il mese di Ramadan.

Bibliografia

api. (s.d.). Tratto da openweathermap.org: http://openweathermap.org/

GOOGLE. (s.d.). */apis*. Tratto da https://code.google.com

GOOGLE. (s.d.). */places/training/autocomplete-android*. Tratto da https://developers.google.com: https://developers.google.com/places/training/autocomplete-android

LatLong. (s.d.). *LatLong Finder*. Tratto da LatLong: http://www.latlong.net/

NASA. (s.d.). *deltaT*. Tratto da eclipse.gsfc.nasa.gov: http://eclipse.gsfc.nasa.gov/SEhelp/deltaT.html

NASA. (s.d.). *JLEX-EU*. Tratto da eclipse.gsfc.nasa.gov: http://eclipse.gsfc.nasa.gov/JLEX/JLEX-EU.html

SajmirRusi. (2014). *redmoon-sajrus*. Tratto da redmoon: redmoon-sajrus.rhcloud.com

Team, A. D. (s.d.). *developer.android*. Tratto da http://developer.android.com/

Weather, O. (2014). *Open Weather*. Tratto da openweathermap.org: http://www.openweathermap.org

wiki. (s.d.). *Luna_rossa_(astronomia)*. Tratto da http://it.wikipedia.org: http://it.wikipedia.org/wiki/Luna_rossa_(astronomia)

api. (s.d.). Tratto da openweathermap.org: http://openweathermap.org/

GOOGLE. (s.d.). */apis*. Tratto da https://code.google.com

GOOGLE. (s.d.). */places/training/autocomplete-android*. Tratto da https://developers.google.com: https://developers.google.com/places/training/autocomplete-android

LatLong. (s.d.). *LatLong Finder*. Tratto da LatLong: http://www.latlong.net/

NASA. (s.d.). *deltaT*. Tratto da eclipse.gsfc.nasa.gov: http://eclipse.gsfc.nasa.gov/SEhelp/deltaT.html

NASA. (s.d.). *JLEX-EU*. Tratto da eclipse.gsfc.nasa.gov: http://eclipse.gsfc.nasa.gov/JLEX/JLEX-EU.html

SajmirRusi. (2014). *redmoon-sajrus*. Tratto da redmoon: redmoon-sajrus.rhcloud.com

Team, A. D. (s.d.). *developer.android*. Tratto da http://developer.android.com/

Weather, O. (2014). *Open Weather*. Tratto da openweathermap.org: http://www.openweathermap.org

wiki. (s.d.). *Luna_rossa_(astronomia)*. Tratto da http://it.wikipedia.org: http://it.wikipedia.org/wiki/Luna_rossa_(astronomia)