JAVA - ANALISI DI DATI SUI PROBLEMI AMBIENTALI

Concentrazioni NO2(g/m3) Annuali e Mensili - Torino Respira

Alunno ***: CAPRARA Silvio, ZHOU Yun Qing***

Classe ***: 4^C Informatica***

Data ***: 10/03/2020***

INDICE

1. PRESENTAZIONE DEL PROBLEMA

“A partire dai siti di ARPA Piemonte o Torino Respira, cercare degli "open data" riguardo a qualità dell'aria o su argomenti che riguardano i cambiamenti climatici o l'ambiente in generale. Cercare i dati in un formato che sia CSV oppure JSON o XML. Comprendere bene il significato dei dati e il loro formato. In seguito, progettare e scrivere un programma JAVA che, tramite interfaccia GUI legga il file con i dati, li elabori e visualizzi in una forma grafica i dati stessi, in modo da poterli interpretare e poi discutere. La lettura da file può essere utilizzata per inserire i vari dati in un vettore (in modalità pila o coda a vostra scelta)".

2. ANALISI DEL FILE .JSON

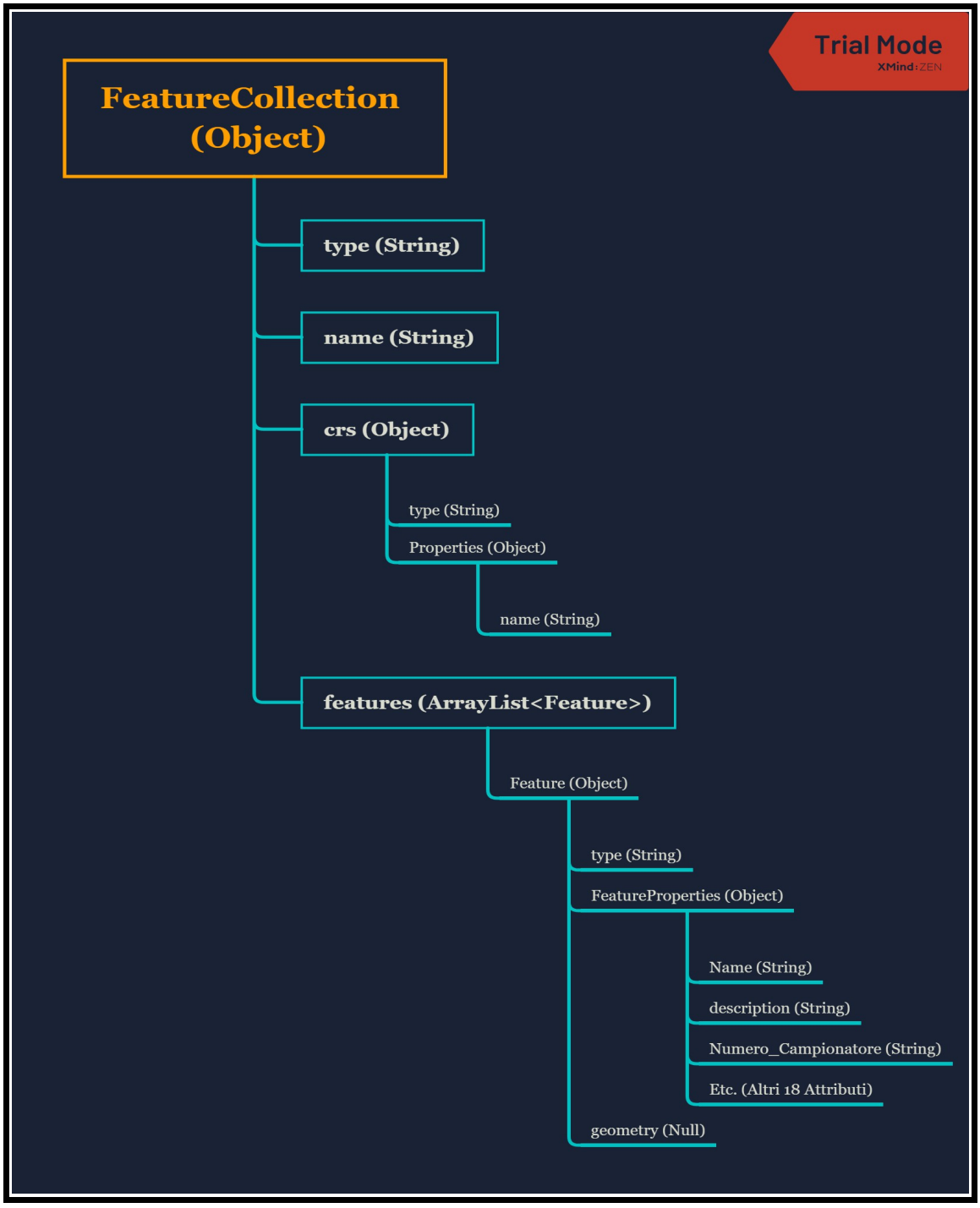
Abbiamo così deciso di prelevare i dati dal sito Torino Respira. Nella sezione #chearitira del sito, è inserita una mappa realizzata con Google My Maps, nella quale sono registrate le misurazioni mensili di NO2 nell'aria (µg/m3). Nelle impostazioni, è stato possibile esportare la mappa come file KML. Tramite dei Tool Online, abbiamo convertito il file **.KML** in **.KMZ**, ed in seguito da **.KMZ** a **.geojson**. Infine, abbiamo validato il file secondo lo standard RFC 8259.


```
{
    "type":"FeatureCollection",
    "name":"Concentrazione NO2 mese µg\/m3",
    "crs":{
        "type":"name",
        "properties":{
            "name":"urn:ogc:def:crs:OGC:1.3:CRS84"
        }
    },
    "features":[
        {
            "type":"Feature",
            "properties":{
                "Name":"Tetti Bertoglio 148, Torino",
                "description":"Numero_Campionatore: 1283092<br>Coord_X: 4025060352<br>Coord_Y: 4992095603
                "Numero_Campionatore":"1283092.0",
                "Coord_X":"4.025060352E9",
                "Coord_Y":"4.992095603E9",
                "Numero_Circoscrizione":"7",
                "Nome_Circoscrizione":"Aurora, Vanchiglia - Sassi - Madonna del Pilone",
                "Comune":"Torino",
                "Via":"Tetti Bertoglio 148",
                "Concentrazione_NO2_mese___g_m3":"22.5",
                "Concentrazione_NO2_anno___g_m3":"18.0",
                "__aumento_ricoveri_x_malattie_respiratorie_mese_oltre_40___g_m3":"0",
                "__aumento_mortalit___generale_anno_oltre_40___g_m3":"0.0",
                "__aumento_sintomi_bronchite_bambini_asmatici_anno_oltre_40___g_":"",
                "Altezza_superiore_a_3m":"N",
                "Cortile":"",
                "Classe_mese":"2.0",
                "Legenda_mese":"20-30",
                "Classe_anno":"1.0",
                "Legenda_anno":"<20",
                "ZTL":""
            },
            "geometry":null
        },
```

Dopo aver convertito il file in un formato leggibile, il passo successivo è stato quello di comprendere la struttura del file stesso, per poi andare a creare le classi Java di appoggio per la lettura. Nei file Json, quando si incontrano delle parentesi graffe '{', siamo in presenza di un dato da trattare come Oggetto. Quando si trovano invece delle parentesi quadre '[', dobbiamo trattare quei dati come una lista (nel nostro caso, abbiamo utilizzato la classe ArrayList).

3. RISOLUZIONE DEL PROBLEMA

Utilizzando la libreria GSON, l'unica complicazione nella lettura del file è stata creare delle classi Java in modo corretto. Abbiamo così ideato uno schema per aiutarci:



Spiegato a parole: Abbiamo una classe principale, FeatureCollection, che andrà a contenere tutti i dati del file. I suoi attributi sono type (Stringa), name (Stringa), crs (oggetto di classe Crs), ed infine features (un ArrayList di Feature).

A sua volta, la classe Crs ha come attributi type (una Stringa), e properties (un oggetto di classe Properties, che a sua volta ha come attributo “name”, ovvero una Stringa).

Analizziamo infine la classe Feature. Quest'ultima ha come attributi type (una Stringa), un oggetto di classe FeatureProperties ed infine “geometry” (sempre impostato a null).

La classe FeatureProperties possiede a sua volta 21 attributi (tutti stringhe).

Dopo aver creato tutto ciò, la classe GSON tramite il metodo

```
<< FeatureCollection featureCollection = gson.fromJson(reader, FeatureCollection.class); >>
```

è in grado di salvare tutto il file di dati all'interno di “featureCollection”.

4. DIAGRAMMA DELLE CLASSI UML

Questo è il diagramma delle classi generato con StarUML, utilizzando l'addon ReverseJava. Il file completo è stato allegato alla consegna.