| **oggetto** | Relazione progetto Programmazione a Oggetti |
| --- | --- |
| **gruppo** | Ribon Luca, mat. 2075517  Bazzan Matteo, mat. 2076422 |
| **titolo** | Parking Manager |

# Introduzione

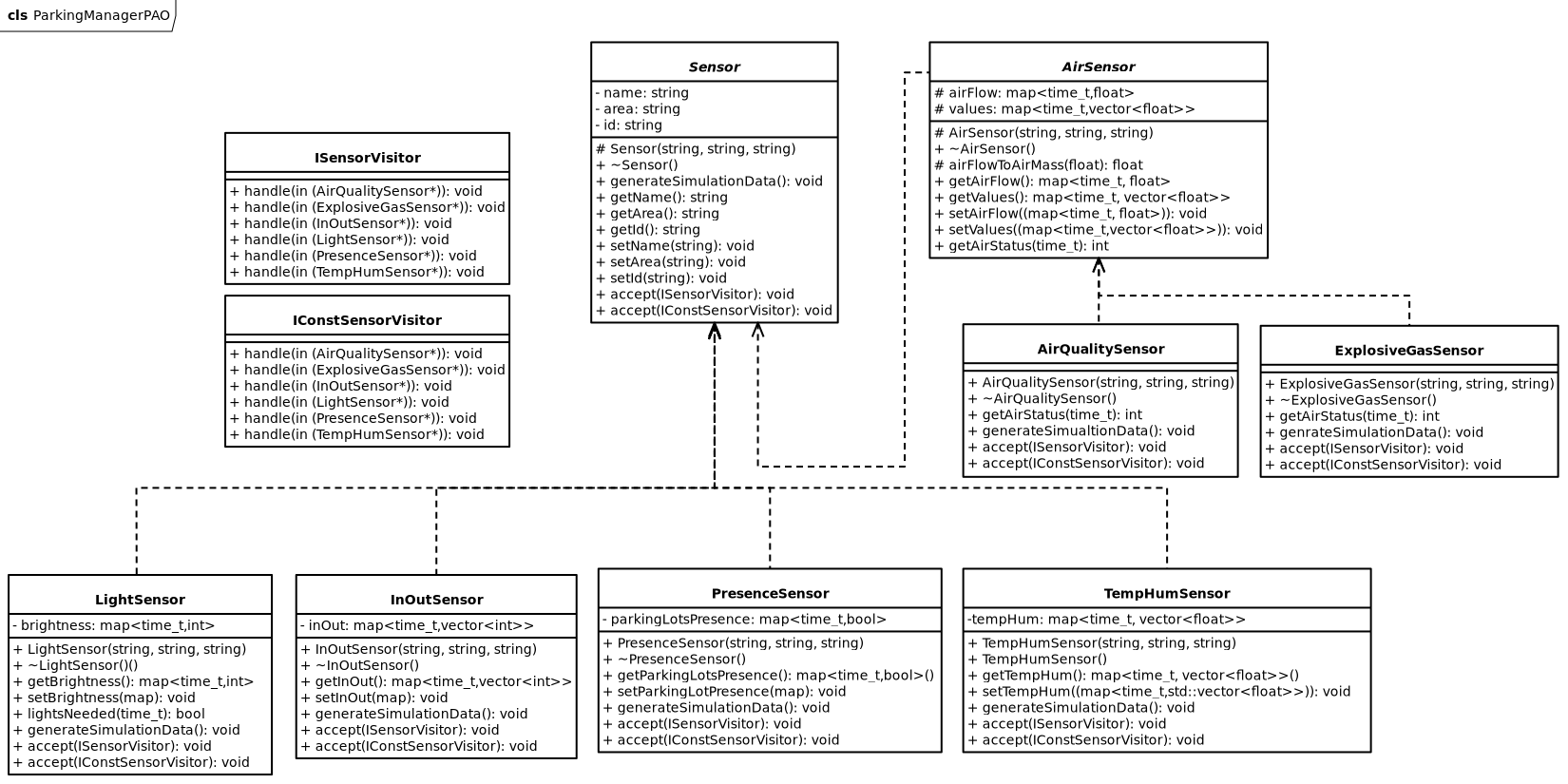
1. Parking Manager è un servizio che permette di gestire un gestire i sensori di un parcheggio, tra i quali:
   * + - 1. - *PresenceSensor*: i sensori di presenza per ogni posto auto
         2. - *LightSensor*: sensori di luminosità
         3. - *InOutSensor*: sensori di entrata e uscita dei veicoli
         4. - *TempHumSensor*: sensori di temperatura e umidità

- *AirQualitySensor*: sensori che analizzano la qualità dell’aria

- *ExplosiveGasSensor*: monitora la presenza di gas potenzialmente esplosivi

* + 1. Le operazioni principali sono:
    2. - aggiunta/rimozione/modifica dei sensori
    3. - visualizzazione dei dati
    4. - visualizzazione dei sensori
    5. - generazione dei dati dei sensori per una simulazione

# Descrizione del modello



Il modello logico è diviso in due parti:

* gestione dei sensori
* gestione della manipolazione dei file JSON usati per caricare e salvare i sensori

*Sensor* è la classe astratta che è la base per tutti gli altri sensori, è caratterizzata da un nome, l’area del parcheggio a cui il sensore appartiene e un id, inoltre presenta le funzionalità *getter* e *setter*, e *generateSimulationData* che permette di generare degli ipotetici valori registrati dai sensori.

Inoltre è presente una seconda classe astratta, figlia a sua volta di *Sensor*, ovvero *AirSensor* che è la base dei sensori che hanno il compito di analizzare l’aria nei pressi del parcheggio; questa classe è caratterizzata da una struttura dati che memorizza le misurazioni del flusso d’aria e il momento in cui il flusso d’aria è stato memorizzato, e un’altra struttura che memorizza la quantità di determinate sostanze nell’aria e il momento a cui avviene ogni misurazione.

Le classi figlie di *Sensor* sono:

* *LightSensor*: il sensore che misura la quantità di luce all’interno del parcheggio nelle varie ore del giorno; può anche indicare se è necessario attivare il sistema di illuminazione o meno
* *InOutSensor*: sensore che registra il numero di veicoli entrati e usciti nelle varie ore del giorno
* *PresenceSensor*: sensore che rappresenta i posti auto e indica se un posto è libero o occupato nei vari momenti della giornata
* TempHumSensor: sensore che misura temperatura e umidità e memorizza l’istante della misurazione

Le classi figlie di *AirSensor* sono:

* *AirQualitySensor*: misura la presenza di sostanza come Co2, No3, Pm10, Pm25 e basandosi su questi dati calcola l’AQI Europeo dando un valore che rappresenta la qualità dell’aria
* *ExplosiveGasSensor*: monitorando la presenza di gas come Metano, Propano, Benzene e Idrogeno, possono restituire un indice di pericolo che si basa sulla percentuale di presenza di questi gas nell’aria

## Polimorfismo

Il polimorfismo viene usato nella generazione dei dati che varia da sensore a sensore, questo avviene tramite il metodo virtuale *Sensor::generateSimulationData();* di cui viene eseguito l’override in ogni classe concreta.

Inoltre il polimorfismo viene usato per la manipolazione dei dati in JSON per far si che vengano salvati correttamenti i sensori, e di conseguenza che vengano creati correttamente i sensori estraendo i dati dal file JSON. Per fare questo è stato implementato il design pattern *Visitor*, tramite le interfacce *ISensorVisitor* e *IConstSensorVisitor*, di cui sono stati implementati i metodi *handle* nella classe *JSONutil* che ha il compito di manipolare i file JSON.

# Persistenza dei dati

*Questa sezione spiega se e come i dati vengono fatti persistere sul file system o altre tecnologie. Non è generalmente necessario entrare nel dettaglio del formato dei file o della struttura delle tabelle SQL, ma è gradita l’aggiunta di un file d’esempio o di uno schema tra i file del progetto, menzionandoli nella relazione.*

Per la persistenza dei dati è stato usato il formato JSON, esso contiene un array di *sensors*. Per ogni oggetto Sensor vengono salvati:

* **id:** identificativo generato tramite l’oggetto *QUuid*
* **name:** nome del sensore
* **area:** area del parcheggio in cui è installato il sensore
* **type:** un intero che grazie alla enum *sensorType* permette di identificare il sensore

Un esempio di file JSON è il file src/Model/Json/2areas.json.

# **Funzionalità implementate**

Io mi sono dedicato all’implementazione del *Model*, quindi:

* la gerarchia delle classi che permettono di gestire 6 tipologie di sensori
* la gestione del polimorfismo
* la generazione dei dati della simulazione
* funzione di ricerca dei sensori
* l'implementazione di salvataggio, lettura e aggiornamento del file JSON

Assieme al mio compagno ho implementato il *Controller* che permette l’interfacciamento tra *View* e *Model*.

La parte relativa della *View* è stata gestita per la maggior parte dal mio compagno, io ho contribuito con qualche aggiunta o risoluzione di bug.

# Rendicontazione ore

| **Attività** | **Ore Previste** | **Ore Effettive** |
| --- | --- | --- |
| Studio e progettazione | 10 | 9 |
| Sviluppo del codice del modello | 20 | 20 |
| Studio del framework Qt | 15 | 17 |
| Sviluppo del codice della GUI | 5 | 4 |
| Test e debug | 5 | 4 |
| Stesura della relazione | 5 | 3 |
| **totale** | 50 | **57** |

Il monte ore è stato superato poiché è stato sottovalutato il tempo da dedicare allo studio del framework Qt e delle librerie JSON con cui nessuno dei due membri del gruppo aveva dimestichezza. Un altro fattore che ha rallentato il progredire del progetto è stato la coordinazione tra i membri del gruppo.   
In fine la progettazione iniziale non è stata affrontata in maniera abbastanza approfondita portandoci ad un risultato finale che non è quello desiderato.