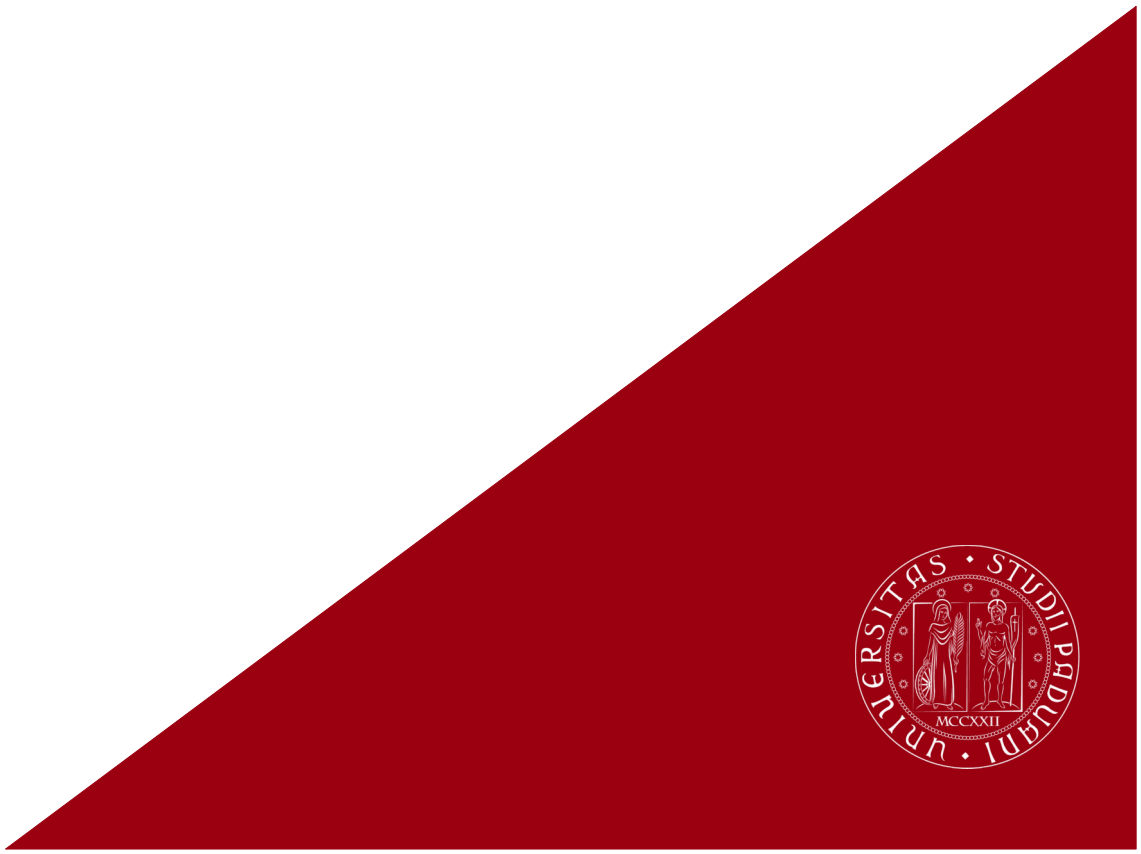
Relazione progetto Programmazione a Oggetti

Titolo: En*g-u-i*ne

Membri del gruppo e numero di matricola:

Piccoli Marco, 2045039

Pivetta Federico, 2009693



Indice generale

Introduzione

Il progetto di Programmazione a Oggetti dell’anno accademico 2023/2024 tratta del seguente tema: i sensori.

Per questo, abbiamo deciso che il nostro lavoro sarà incentrato sulla creazione di *Eng-u-ine*.

Eng-u-ine è una applicazione che permette, ai normali utenti, la visualizzazione di sensori che fanno riferimento allo stato di ogni singolo componente all’interno di una automobile visualizzando, per ognuno, un grafico che mostra l’andamento di tale stato negli ultimi 12 mesi.

L’utente all’interno di *Eng-u-ine* può personalizzare a suo piacimento i sensori inserendone di nuovi con all’interno dei campi preimpostati, ma anche modificare quelli già esistenti oppure eliminarli.

L’utente in fase di creazione dei sensori può scegliere su una lista di 5 sensori:

- sensore *Batteria*: è un sensore che monitora il tempo di ricarica della batteria nel tempo;
- sensore *Consumo*: è un sensore che monitora i consumi del veicolo nel tempo;
- sensore *Gas*: è un sensore che monitora i gas prodotti dai tubi di scarico per rendere consapevole l’utente dell’impatto ambientale;
- sensore *Motore*: è un sensore che monitora la composizione e la corretta accensione del motore;
- sensore *Pneumatico*: è un sensore che rileva l’usura dei pneumatici con il passare dei mesi/anni

Descrizione del modello

Il modello logico di questo progetto si divide in due parti: la prima parte comprende la gestione dei vari sensori partendo da una classe astratta *Sensor* e le sue sottoclassi come si nota in Figura 1, mentre la seconda comprende l'interfaccia che andrà ad implementare le funzionalità come la ricerca, la creazione, la modifica e l'eliminazione di tali *Sensor*.

Alla classe astratta *Sensor* e alle sue sottoclassi vengono aggiunte delle classi di servizio per convertire, in formato JSON (e viceversa), e salvare su file i risultati ottenuti dai diversi sensori. Per tale scopo si è deciso di utilizzare gli strumenti offerti da *Qt*, in particolare il *QjsonObject* .

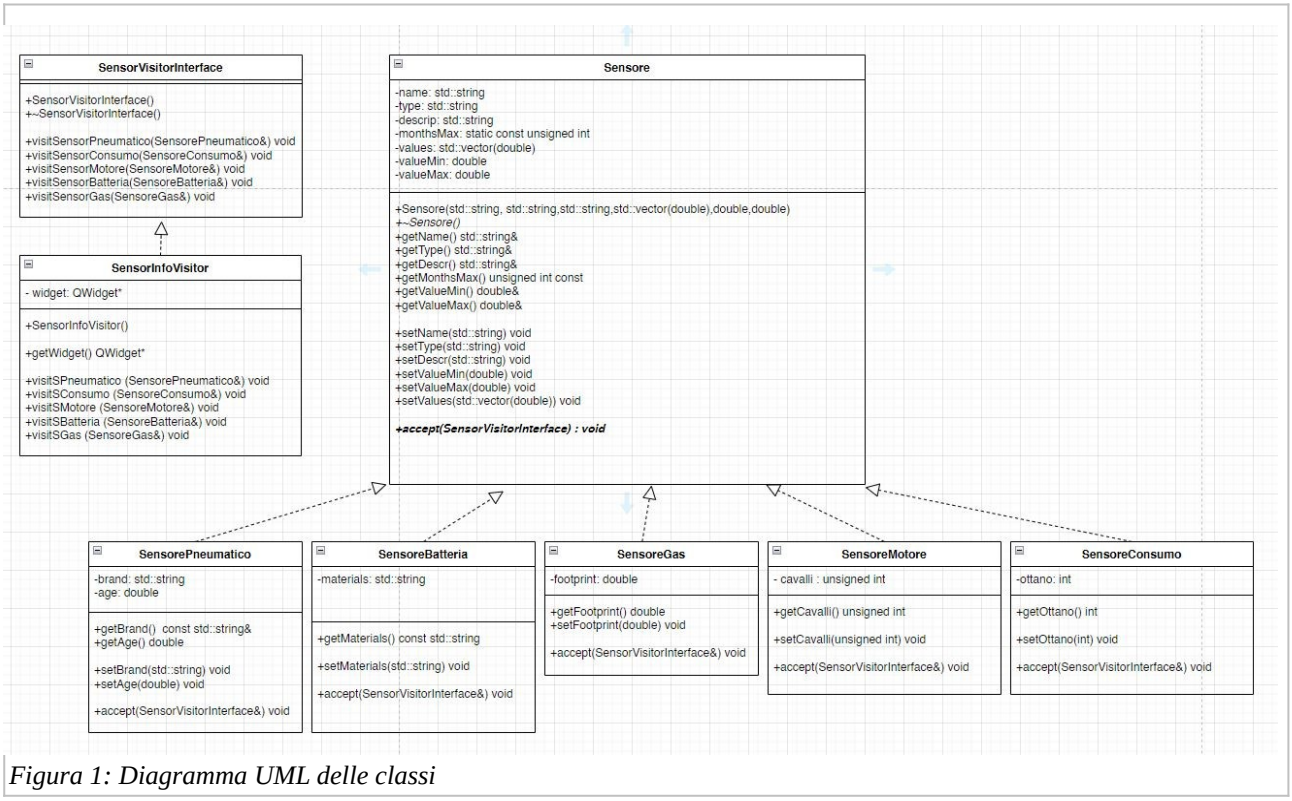


Figura 1: Diagramma UML delle classi

Tutta la struttura gerarchica parte dalla classe astratta *Sensor* che, al suo interno, contiene tutte le informazioni comuni ai sensori. Tali informazioni sono:

- nome
- tipologia di sensore
- descrizione
- un vettore contenente i valori del sensore
- il valore minimo e il valore massimo di tale vettore

Per ognuno di questi campi vengono implementati i metodi *getter* e *setter*.

Da *Sensor* vengono definite cinque classi concrete che rappresentano, come descritto nell'Introduzione, i sensori veri e propri specificando, per ognuno di essi, dei campi e delle classi aggiuntive per differenziarne l'utilizzo.

I sensori si dividono in cinque categorie:

- *Motore*: viene aggiunto il campo *cavalli* che indica il numero di CV che il motore può produrre a pieno regime;
- *Gas*: viene aggiunto il campo *footprint* che indica l'impatto ambientale che ha il carburante utilizzato nell'automobile;
- *Consumo*: viene aggiunto il campo *ottano*, che indica il numero di ottani che ha l'idrocarburo consumato (es. diesel = 25, benzina = 95 etc…);
- *Batteria*: viene aggiunto il campo *materials*, che indica con che materiale è composta la batteria della macchina (es. litio, nichel etc…);
- *Pneumatico*: vengono aggiunti i campi *brand* e *age* che indicano, rispettivamente, la marca e gli anni del / dei pneumatici montati.

Per ognuno dei campi aggiunti vengono implementati i metodi *getter* e *setter*.

Poiché ogni sensore specifico all'interno del modello si comporta come un ***D**ata **T**ransfer **O**bject* (DTO) e non dispongono di nessuna funzione rilevante, abbiamo scelto di utilizzare il design pattern dei *Visitor* per consentirne l'arricchimento in maniera dinamica.

Per questo scopo abbiamo realizzato la classe *SensorVisitorInterface* che si comporta da interfaccia per tutti i *Visitor* a cui vengono aggiunti tutti i metodi virtuali puri; tali metodi vengono implementati da due classi: *SensorInfoVisitor* e *SensorModifyVisitor*. La differenza sostanziale tra le due classi, come si può intuire facilmente, è l'utilizzo a cui vengono sottoposte infatti la prima implementa tutti i metodi per ricavare le informazioni di un certo sensore, mentre la seconda implementa i metodi che servono a modificare i dati di un qualsiasi sensore quando richiesto dall'utente. Di conseguenza, in *Sensor* viene aggiunto il metodo virtuale puro *accept* che viene implementato da ogni singolo sensore per poter accettare i due tipi di *Visitor*.

Polimorfismo

Persistenza dei dati

Funzionalità implementate

Rendicontazione ore