# Relazione Progetto: CarSensor

Andrea Perozzo(2082849)

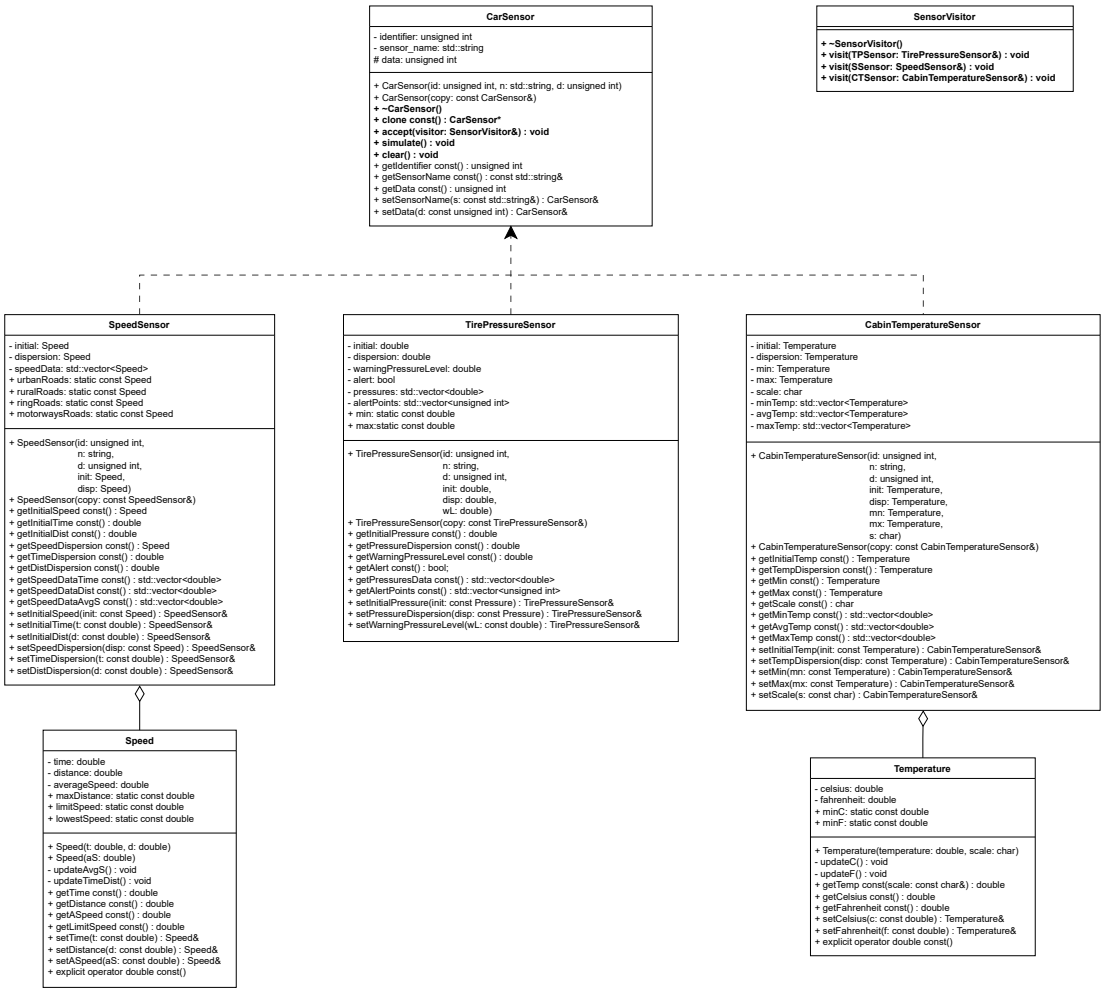
## 1 Introduzione

Il progetto CarSensor è progettato per monitorare e registrare tre specifici parametri di un auto. I sensori implementati sono i seguenti:

- **CabinTemperatureSensor**: mostra il range di temperatura nell'abitacolo, tra un valore minimo e uno massimo in base alla temperatura impostata. Può essere misurata in:
  - Celsius
  - Fahrenheit
- **SpeedSensor**: questo sensore mostra la velocità media dei vari tragitti effettuati, indicando quale strada sia stata maggiormente percorsa durante il tragitto, potendo osservare sull'asse delle ordinate del grafico i limiti correlati alle seguenti strade:
  - Centro abitato (50 km/h)
  - Strada extraurbana (90 km/h)
  - Tangenziale (110 km/h)
  - Autostrada (130 km/h)
- **TirePressureSensor**: definisce la pressione degli pneumatici e i livelli di allerta secondo i quali la pressione risulta inferiore ad un determinato valore soglia impostato dall'utente

## 2 Descrizione del modello

Il modello logico dei sensori è organizzato in una gerarchia di classi che gestisce la creazione, la distruzione, e la generazione dei dati, oltre a includere diversi metodi getter e setter per leggere e modificare i vari dati dei sensori. Il diagramma sottostante illustra la gerarchia delle classi, progettata in conformità con le tre concretizzazioni richieste dai vincoli del progetto.




Alla base della gerarchia si trova la classe astratta "CarSensor", che include gli attributi e i metodi condivisi da tutti i tipi di sensori. Gli attributi comuni comprendono l'ID, il nome del sensore e un attributo chiamato "data" che rappresenta il numero di campionamenti. I metodi comuni sono il distruttore, il metodo clone (per clonare un sensore), accept (utilizzato per il salvataggio dei sensori), simulate (per generare i dati e riempire i vettori corrispondenti), e clear (per svuotare i vettori contenenti i dati).

# 3 Polimorfismo

L'utilizzo principale del polimorfismo avviene tramite il design pattern del Visitor.
Il componente **ChartInfo** mostra le varie informazioni del sensore e il grafico della simulazione. Il Visitor di **ChartInfo** visita il sensore per determinarne la tipologia e riempire i campi in base alle

caratteristiche rilevate. Inoltre, **ChartInfo** è responsabile dell'invocazione del metodo 'simulate()' del sensore, il quale raccoglie i dati necessari per inserirli nel grafico che viene generato sfruttando il Visitor presente in **Chart**.
Il componente **SensorsPanel** rappresenta la lista dei sensori disponibili. Utilizza un Visitor che determina il tipo dinamico del sensore e visualizza un widget con l'icona corrispondente, oltre ai dati di ID e nome del sensore.

# 4 Persistenza dei dati

Per la memorizzazione dei dati viene utilizzato il formato JSON, che contiene un vettore di oggetti sensori. Questi oggetti sono principalmente strutturati come coppie chiave-valore, e la serializzazione delle sottoclassi viene gestita aggiungendo un attributo "type". La gestione del JSON introduce un ulteriore livello di gestione dei dati, distinto del vettore presente nella MainWindow. All'interno della classe JsonRepository, viene creata una std::map che clona i sensori presenti nel vettore, per poi inserirli nel file JSON. Successivamente, JsonRepository si occupa di recuperare i dati dal JSON, clonandoli nuovamente per copiarli nel vettore della MainWindow.
Un esempio di persistenza dei dati è il file Example.json presente nella cartella del progetto.

# 5 Funzionalità implementate

Una volta avviata l'applicazione, è possibile aggiungere un nuovo sensore visibile sul pannello, mediante l'utilizzo di uno dei primi tre pulsanti disponibili nella ToolBar: "Add Cabin Temperature Sensor," "Add Speed Sensor," e "Add Tire Pressure Sensor." Durante l'inserimento, l'utente può specificare l'ID, il nome, data e gli altri dati del sensore (con l'ID come unico campo obbligatorio, non è possibile creare due sensori con ID uguali).
Dopo aver inserito i dati per quelo specifico sensore, cliccandoci sopra, è possibile visualizzare i dettagli dello stesso e consultare il grafico corrispondente premendo il tasto "Simulate" in alto a sinistra. Il pulsante "Edit" consente di modificare i dati del sensore, mentre il pulsante "Delete" nel widget permette di eliminarlo.
Sopra il pannello dei sensori è presente una casella di testo per la ricerca: inserendo il nome o l'ID di uno specifico sensore, si può effettuare la ricerca all'interno dell'elenco. Inoltre, nella Toolbar, la funzionalità "Save as" consente di salvare i sensori aggiunti in un file '.json', mentre la funzionalità "Open" permette di aprire un file '.json' precedentemente salvato.
Le operazioni di "Open" e "Save as" sono accessibili anche tramite le shortcut da tastiera 'Ctrl+O' e 'Ctrl+S', rispettivamente.

# 6 Rendicontazione ore

Questa è stata la mia prima esperienza con la programmazione grafica, durante la quale ho sviluppato il progetto. Dato che l'ambiente di sviluppo Qt era per me del tutto nuovo, ero consapevole che avrei impiegato un tempo considerevole per portarlo a termine, superando di gran lunga le 50 ore di lavoro. Ho iniziato il progetto il 18 luglio e l'ho concluso il 20 agosto. Di seguito è riportata una ripartizione delle attività svolte, con le ore previste e quelle effettivamente dedicate a ciascuna di esse.

| **Attività** | **Ore previste** | **Ore effettive** |
|---|---|---|
| Studio e progettazione | 10 | 10 |
| Sviluppo del codice del modello | 10 | 20 |
| Studio del framework Qt | 20 | 20 |
| Sviluppo del codice della GUI | 20 | 30 |
| Test e debug | 10 | 10 |
| Stesura della relazione | 5 | 5 |
| **Totale:** | **75** | **95** |