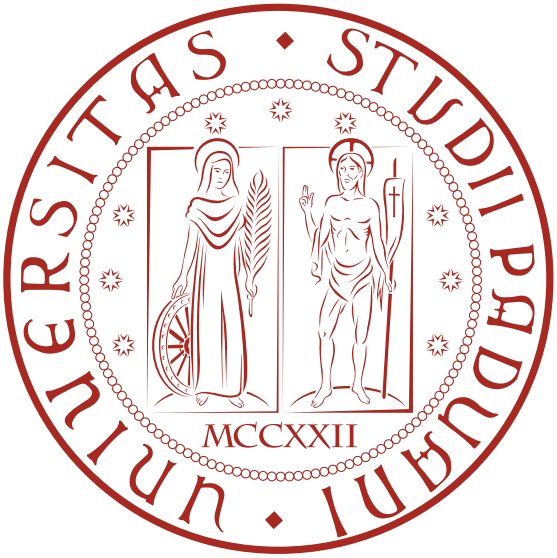


Realizzazione di un simulatore di sensori

Corso di programmazione a oggetti

Anno Accademico 2023-2024

Indice

1 Introduzione

1.1 Sintesi del programma

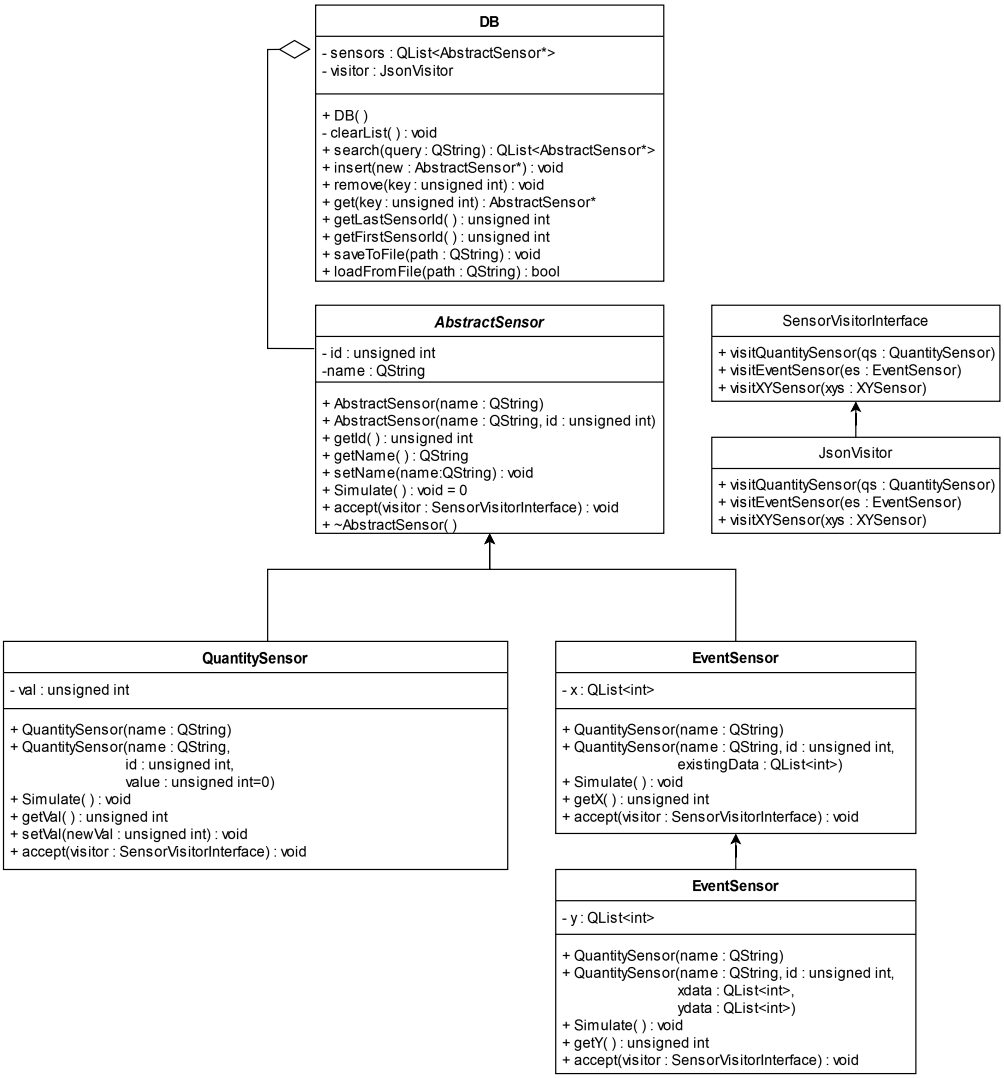
Sensori è un programma scritto in C++ e implementa il framework Qt per presentare le informazioni all'utente attraverso una GUI. Il compito dell'applicazione è quello di gestire delle simulazioni di sensori e salvarle in maniera persistente sul disco. Il programma mette a disposizione tre tipi di sensori: quelli che rilevano una quantità (`QuantitySensor`), quelli che rilevano quanti eventi ci sono stati in una settimana(`EventSensor`) e quelli che rilevano un valore e la sua variazione nel tempo (`XYSensor`). Questa scelta è stata fatta in modo tale da utilizzare tre tipi di grafici:

- Una barra del progresso per rappresentare i `QuantitySensor`;
- Un grafico a barre per rappresentare gli `EventSensor`;
- Un grafico lineare per rappresentare gli `XYSensor`.

L'interfaccia grafica del programma è composta da due componenti principali, il **browser** che ha il compito di elencare tutti i sensori presenti e l'**inspector** che si occupa della gestione e della simulazione del sensore selezionato.
Questo progetto offre il supporto persistenza dei dati tramite l'uso del formato strutturato Json.

2 Descrizione del modello

Il programma è basato sull'architettura view-model e questa sezione tratterà delle scelte implementative del model, ovvero sensori, struttura dati e il loro salvataggio sul disco. Il seguente diagramma UML rappresenta le relazioni tra le classe del namespace `model`.



2.1 La gerarchia di sensori

Il codice dei sensori si trova all'interno del namespace `model::sensor`.
Ogni classe ha due costruttori, uno che inizializza dei sensori nuovi (creati dall'utente) e uno per i sensori già creati in precedenza (importati da Json).
Il primo elemento che compone la gerarchia è la classe astratta `AbstractSensor`. Questo è composto da un identificatore univoco generato casualmente e da un nome. Oltre ai metodi accessori sono presenti il metodo virtuale puro `simulate()` e il metodo `accept(SensorVisitorInterface)` necessario per implementare il pattern Visitor2.3.
La prima classe derivata è `QuantitySensor` e si occupa di salvare un solo valore intero nell'intervallo [0,1000] (ad esempio una bilancia). L'intervallo è stato deciso arbitrariamente e si trova in `QuantitySensor::simulate()`.
La seconda classe derivata da `AbstractSensor` è `QuantitySensor`. Il suo compito e quello di

misurare quante volte al giorno accade un certo evento nell'arco di una settimana. Si è scelto di usare una `QList`, quindi una struttura che per sua natura non ha una dimensione fissa allocata perché verrà riutilizzata dalla classe sottostante con dimensione diversa.
Il terzo tipo di sensore è derivato da `EventSensor` e si chiama `XYSensor`. Questa classe aggiunge un'ulteriore `QList` a quella già presente in modo da creare dei punti del tipo (x[i], y[i]) da inserire nel grafico.

2.2 La struttura dati

La gestione dell'insieme di sensori è affidata alla classe `model::database::DB`. I due attributi privati della classe sono una `QList<AbstractSensor>` e un `JsonVisitor`. Ho scelto di allocare il `JsonVisitor` alla creazione dell'oggetto `DB` perché leggermente più efficiente di crearlo ad ogni invocazione di `DB::saveToFile()`.
Il metodo `DB::search(String query)` prende in input l'id oppure il nome (anche parziale) di un sensore e ritorna una `QList<AbstractSensor>` di sensori compatibili con l'input della ricerca; se la stringa è vuota ritorna tutto il database. Questo metodo viene invocato dalla GUI ogni volta che si digita qualcosa nel **browser** ed ha una complessità O(n). Ogni volta che si clicca un sensore il suo id viene passato all'**inspector** che tramite il metodo `AbstractSensor* DB::get(unsigned int key)` riceve il suo puntatore e usa il Visitor per mostrarlo come descritto nel capitolo sul polimorfismo 2.3. I metodi `getFirstSensorId()` e `getLastSensorId()` servono per selezionare un sensore rispettivamente dopo la cancellazione di un altro sensore o la creazione di uno nuovo. I metodi `saveToFile(String path)` e `loadFromFile(String path)` sono spiegati nel capitolo sulla persistenza dei dati 2.4.

2.3 Polimorfismo

Questo progetto sfrutta il polimorfismo in tre punti: per la simulazione, per ricavare il giusto grafico e per la generazione del Json durante l'esportazione dei dati.
La simulazione avviene tramite il metodo virtuale `void AbstractSensor::simulate()` che è stato poi ridefinito dalle classi sottostanti in modo da adattarlo alla struttura dati del sensore.
L'utilizzo principale del polimorfismo riguarda l'implementazione del design pattern Visitor in due casi distinti.
La prima istanza del pattern la si trova nella classe `SensorVisitor`, utilizzata in `SensorInspectorWidget`, entrambe contenute nel namespace `view::sensorViewer`. Il suo compito è quello di costruire il giusto `QWidget` in base al sensore selezionato dall'utente. I tipi di grafico che `SensorVisitor` può ritornare sono tre:

- Una `QProgressBar` (barra di caricamento) con degli elementi di CSS personalizzati per rappresentare le istanze di `QuantitySensor`;
- Un `QChart` che contiene un `QBarSet` (grafico a barre) incapsulato in un `QChartView` (che eredita da `QWidget`) per rappresentare gli `EventSensor`;
- Un `QChart` (grafico cartesiano) incapsulato in un `QChartView` per rappresentare gli `XYSensor`.

La seconda istanza del Visitor è contenuta nella classe `model::Json::JsonVisitor` ed è dedicata alla costruzione dei vari oggetti `QJsonObject` utilizzati per la persistenza dei dati, trattata nel capitolo apposito2.4.
Per implementare questo design pattern è stato necessario creare l'interfaccia `SensorVisitorInterface` e farla implementare alle classi `SensorVisitor` e `JsonVisitor`. Infine è stato necessario aggiungere il metodo `void accept(SensorVisitorInterface)` alla classe `AbstractSensor` e alle relative sottoclassi.

2.4 Persistenza dei dati

La persistenza dei dati è gestita tramite file Json in quanto nativamente supportato da Qt e facilmente leggibile usando un qualsiasi editor di testo. Il programma permette di importare ed esportate tutta la lista di sensori dal menù `File` o usando gli shortcut indicati tra parentesi.
L'elemento principale di ogni file è un array di oggetti sensore. Ogni oggetto è composto da `id`, `name`, `type` e i dati relativi all'ultima simulazione che in questo caso possono essere un singolo valore, uno o due array.
L'import è gestito dal metodo `DB::loadFromFile(QString path)` che viene invocato dalla `MainWindow`. Questo metodo, dato un percorso apre il file, controlla se il Json è ben formato, importa solo i sensori che contengono i campi corretti e infine sostituisce la lista di sensori precedente con quella appena creata.
L'export, invece, viene gestito dal metodo `DB::saveToFile(QString path)`, anch'esso invocato dalla `MainWindow`. Questo metodo, sfruttando la classe `JsonVisitor`, che implementa il Visitor design pattern, crea un `QJsonObject` per ogni sensore e lo inserisce nel `QJsonArray` principale che diventerà poi la radice del file.

3 Funzionalità implementate

Le funzionalità implementate nel programma sono suddivise in base alla categoria.
Per la gestione di un singolo sensore ho implementato le seguenti funzionalità:

- Creazione attraverso un wizard che controlla la correttezza dei campi;
- Modifica del nome;
- Simulazione;
- Cancellazione;
- La visualizzazione di un `QuantitySensor` fa uso di elementi CSS personalizzati per cambiare l'aspetto del widget;

Per la gestione dell'insieme di sensori:

- Barra di ricerca in tempo reale per filtrare i sensori attraverso il nome o l'ID;
- Importazione ed esportazione di una configurazione di sensori e dei log dell'ultima simulazione attraverso un file Json. Sono state gestite le seguenti casistiche:
  - Se il file non è ben formato si riceve un warning e viene annullata la procedura;
  - Se un sensore non ha gli attributi richiesti viene ignorato (ad eccezione del `QuantitySensor` che viene impostato di default a zero);
  - Se provo ad esportare una configurazione vuota si riceve un warning e non si può procedere;
- Shortcut da tastiera per richiamare le funzionalità del menù 'File' (i tasti da premere sono scritti tra parentesi).

Funzionalità generiche dell'interfaccia:

- La GUI fa uso della libreria QtAwesome[1] che permette in modo semplice e rapido di inserire icone di tipo `QIcon`, quindi nativamente supportate;
- Le due parti dell'interfaccia, ovvero il browser e l'inspector sono contenute in un `QSplitter` che permette di ridimensionarle posizionando il cursore tra i due elementi;
- Avviso in caso di chiusura del programma con sensori presenti;

4 Rendicontazione ore

Le ore effettive non sono da intendersi come consecutive (o completamente corrispondenti al vero) in quanto sono più volte tornato indietro a modificare le scelte architetturali.

Fase progettuale	**Ore previste**	**Ore effettive**
Lettura e creazione appunti sulle specifiche del progetto	2	1
Progettazione dei tipi di sensore e della struttura dati	3	5
Progettazione del wireframe e del flowchart dell'applicazione	4	5
Consultazione della documentazione di Qt e dei progetti esempio	6	5
Implementazione del model	7	8
Implementazione della GUI	15	20
Implementazione della persistenza dei dati tramite Json	4	3
Stesura documentazione[2]	4	9
Totale	**45 ore**	**57 ore**

5 Compilazione ed esempi Json

Per compilare ed eseguire il programma è necessario aprire il terminale ed eseguire i seguenti comandi:

```
cd src
qmake
make
./Sensori
```

Nella cartella `esempi_json` sono presenti tre file di configurazione di sensori:

- `valid.json` non presenta anomalie;
- `ignored_invalid_sensor.json` ha il secondo sensore senza tipo e il terzo senza dati;
- `invalid_warning.json` contiene un Json malformato e si riceverà un warning come descritto nella sezione sulla persistenza 2.4.

[1] https://github.com/gamecreature/QtAwesome

[2] Sono state necessarie più ore del previsto in quanto ho cercato di installare LaTeX sia su Windows che su Linux. Per via di problemi con la compilazione e l'installazione dei pacchetti ho poi scelto di utilizzare Overleaf.