Progetto di Programmazione ad oggetti

QTo-do

Damiano Zanardo (1193216)

Sommario

QTo-do è un'applicazione desktop che permette la gestione di semplici *to do*, *memo*, *reminder* ed alcune combinazioni, ad esempio *to do* + *reminder* e *memo* + *to do*.

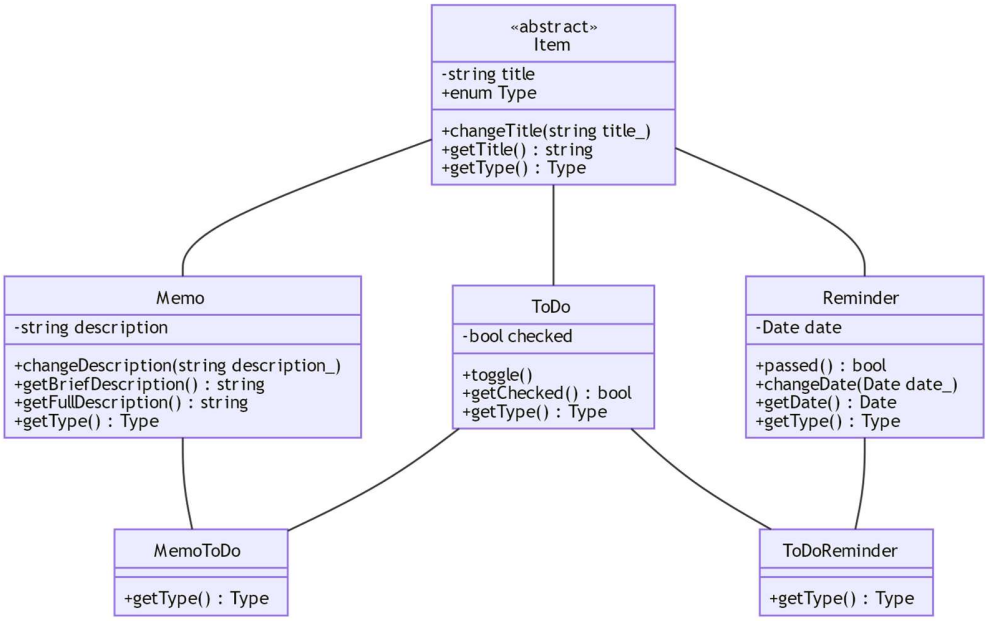
Gruppo formato da:

Damiano Zanardo (1193216)
Riccardo Pavan (1189938)

1 Gerarchie

QTo-do comprende due principali gerarchie: una riguardante il modello mentre l'altra riguardante la GUI.

1.1 Gerarchia Modello



Item è la classe base astratta che offre la gestione del campo `title` (con `changeTitle()` e `getTitle()`) e un metodo virtuale puro `getType()` che ritorna un valore da `Type` a seconda del tipo di *item* che si sta gestendo.

ToDo è una classe concreta derivata da `Item` che permette la gestione del campo `checked` (con `toggle()` e `getChecked()`) per aggiungere la possibilità di spuntare i to do. La classe implementa il metodo virtuale puro `getType()` ritornando `Item::Type::ToDo`.

Memo è una classe concreta derivata da `Item` che permette la gestione del campo `description` (con `changeTitle()`, `getBriefDescription()` e `getFullDescription()`) per aggiungere la possibilità di avere una descrizione. La classe implementa il metodo virtuale puro `getType()` ritornando `Item::Type::Memo`.

Reminder è una classe concreta derivata da `Item` che permette la gestione del campo `date` (con `changeDate()`, `getDate()`) per aggiungere la possibilità di avere una data. Il metodo `passed()` serve per capire se un reminder è scaduto o no. Per il campo `date` è stata creata una classe **Date** perché lo standard non fornisce una classe semplice da utilizzare per lo scopo. La classe implementa il metodo virtuale puro `getType()` ritornando `Item::Type::Reminder`.

Queste prime 3 classi sono tutte derivate virtualmente da `Item` dato che serviranno per l'ereditarietà a diamante.

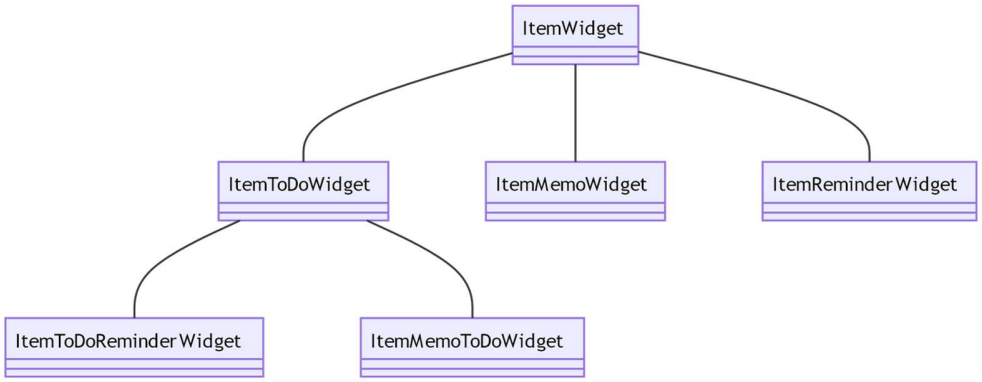
ToDoReminder è una classe concreta derivata da `ToDo` e `Reminder`. La classe implementa il metodo virtuale puro `getType()` ritornando `Item::Type::ToDoReminder`.

MemoToDo è una classe concreta derivata da `Memo` e `ToDo`. La classe implementa il metodo virtuale puro `getType()` ritornando `Item::Type::MemoToDo`.

Queste due classi offrono soltanto la combinazione delle due classi da cui derivano.

Il contenitore degli oggetti del modello è un `MyVector`, un contenitore che si comporta come un *vector*.

1.2 Gerarchia GUI



Questa gerarchia offre dei widget (che saranno visualizzabili in una lista) per i vari tipi di oggetti istanziabili dal modello

`ItemWidget` è la classe base astratta derivata da `QAbstractButton` per avere il controllo del click sui widget stessi. Il metodo virtuale puro è uno *slot* `onItemSelected()` che viene connesso al *signal* `clicked()` di `QAbstractButton`. Una label visualizzerà il titolo dell'item.

Le 3 classi **`ItemToDoWidget`**, **`ItemMemoWidget`** e **`ItemReminderWidget`** sono i widget per visualizzare rispettivamente i to do, i memo e i reminder. In particolare, `ItemToDoWidget` ha una checkbox per spuntare o meno un to do già fatto, mentre `ItemMemoWidget` ha una label per visualizzare la descrizione in breve e `ItemReminderWidget` ha una label per visualizzare la data di scadenza del reminder.

Siccome gli oggetti derivati da `QObject` non supportano la derivazione virtuale è stato necessario derivare **`ItemToDoReminderWidget`** e **`ItemMemoToDoWidget`** solamente da **`ItemToDoWidget`** (e quindi non da `ItemReminderWidget` e `ItemMemoWidget`). Così facendo si ottiene la gestione della checkbox reimplementato il metodo `strikethrough()` per barrare l'item quando è stato spuntato.

Per contenere questi *widgets* è stato utilizzato un `QButtonGroup`, una classe che permette la gestione di una lista di `QAbstractButton` come se fossero dei *radio-button*, così da poter selezionare un solo oggetto alla volta. Ogni *button* della lista di `QAbstractButton` ha un *id* che è stato impostato per essere concorde con l'index della lista del modello. `QButtonGroup` fornisce dei metodi utili come `checkedId()` che ritorna l'id del *button* selezionato e quindi anche l'index dell'item nel `MyVector`.

1.3 Descrizione delle chiamate polimorfe

Nella classe **`Item`** sono presenti i seguenti metodi virtuali puri:

- `virtual ~Item() = default;`
- `virtual Item::Type getType() const = 0;`

    - che viene reimplementato in tutte le classi derivate per ritornare l'*enum* `Type` corretto a seconda del tipo di oggetto.

Nella classe `ItemWidget` è presente il seguente motodo (*slot*) virtuale puro:

- `virtual void onItemSelected() = 0;`
    - che viene reimplementato in tutte le classi derivate per emettere il *signal* `itemSelected(Item::Type)` corretto a seconda del tipo di widget cliccato.

Nella classe ItemToDoWidget è presente il seguente motodo virtuale:

- `virtual void strikethrough();`
    - che di default barra il titolo quando la checkbox è spuntata e viene reimplementato in `ItemToDoReminderWidget` e `ItemMemoToDoWidget` per barrare rispettivamente la data e la descrizione quando la checkbox è spuntata.

2 ItemSender

La funzione `void itemSender(Item * item_, int index_, SendTo edit)` serve per emettere le informazioni degli item del contenitore alla GUI tramite *signals*. La funzione controlla (`SendTo edit`) a che parte della GUI deve inviare le informazioni e, in base al tipo di item, invia segnali diversi.

I segnali inviati a `ListItemWidget` (il widget che contiene la lista `QAbstractButton`[1]) sono uno per tipo di item (es.: `Item::Type::ToDo` → `sendToDoInformation(int, QString, bool)`) perché sarà questo widget a creare gli `ItemWidget` da inserire nella lista; es.: `addItemWidget(index_, new ItemToDoWidget(title_, checked_))`.

I segnali inviati alla parte di modifica invece sono più generali dato che non è necessario costruire dei widget ma soltanto aggiornare dei campi di input come `QLineEdit` (*title*), `QTextEdit` (*description*) oppure `QDateEdit` (*date*): `sendTitleInformation(QString)`, `sendDescriptionInformation(QString)`, `sendDateInformation(QDate)`.

3 Descrizione di I/O

Il programma carica e salva gli item all'interno di un file XML (`./data.xml`). È stato scelto il formato XML perché è abbastanza intuitivo, facile da convertire in altri formati e inoltre QT fornisce le classi `QXmlStreamReader` e `QXmlStreamWriter` che permettono la lettura e la scrittura di file XML.

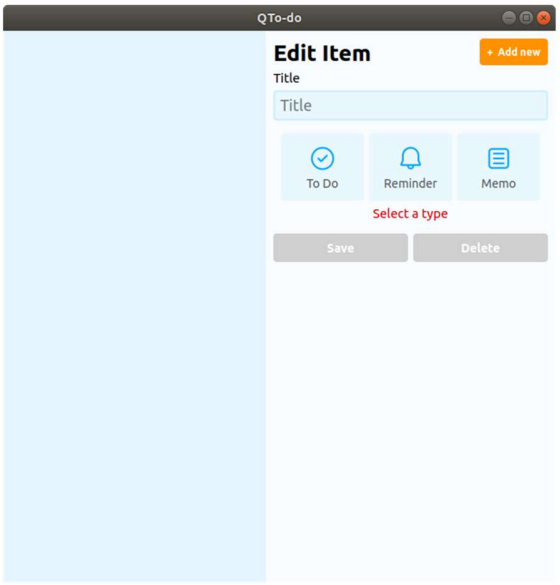
Esempio di file `./data.xml`:

[1] Vedi *1.2 Gerarchia GUI*

```
<Items>
    <Item type="ToDo">
        <title>Comprare gelato</title>
        <checked>false</checked>
    </Item>
    <Item type="ToDoReminder">
        <title>Consegna progetto</title>
        <checked>true</checked>
        <date>gio set 3 2020</date>
    </Item>
    <Item type="Reminder">
        <title>Inizio lezioni</title>
        <date>lun set 28 2020</date>
    </Item>
    <Item type="Memo">
        <title>Come scrivere &amp;quot;ciao&amp;quot;</title>
        <description>Si preme la &amp;quot;c&amp;quot; poi la &amp;quot;i&amp;quot; poi la &amp;quot;a&amp;quot; poi la &amp;quot;a&amp;quot;.
In questo modo si scrive ciao.</description>
    </Item>
    <Item type="Reminder">
        <title>Natale</title>
        <date>mer dic 25 2019</date>
    </Item>
</Items>
```

Nel controller sono stati implementati due metodi: `loadXML()` e `saveXML()` che utilizzano rispettivamente `QXmlStreamReader` e `QXmlStreamWriter`. Come scritto nel *Manuale utente della GUI* il salvataggio avviene automaticamente dopo aver creato/modificato/eliminato un item dal contenitore.

4 Manuale utente della GUI



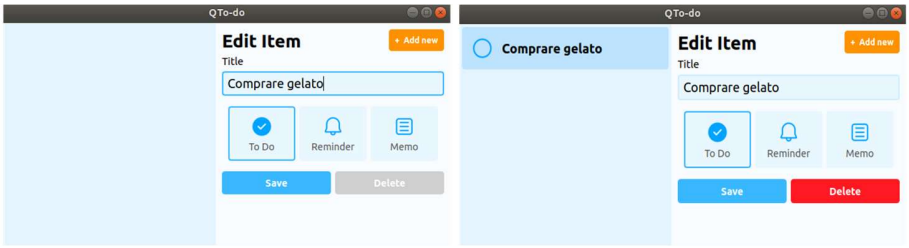
Questa è la schermata iniziale della GUI. A sinistra ci saranno i vari item. Mentre a destra c'è la sezione per creare e modificare gli item.

Il pulsante *+ Add New* serve per iniziare la creazione di un nuovo item.

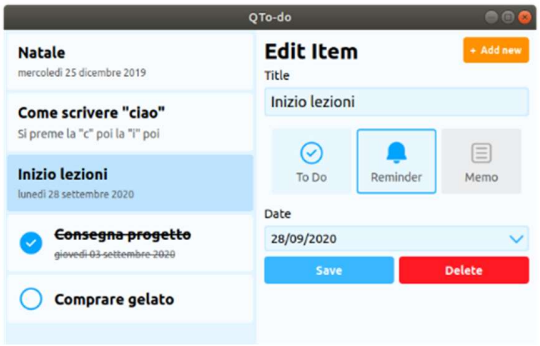
I tre pulsanti *To Do*, *Reminder* e *Memo* servono per selezionare il tipo di item da creare oppure cambiare il tipo dell'item selezionato.

Selezionando *Reminder* è possibile scegliere la data di scadenza del reminder.

Selezionando *Memo* è possibile inserire il testo della descrizione.

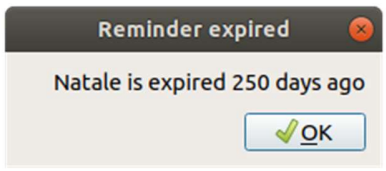


Cliccando *Save* viene salvato l'item e mostrato nella parte sinistra.



Selezionando un item da sinistra si vedono i dettagli nella parte di destra ed è possibile modificare le informazioni o il tipo di item. Selezionando un item e successivamente cliccando *Delete* si elimina l'item selezionato.

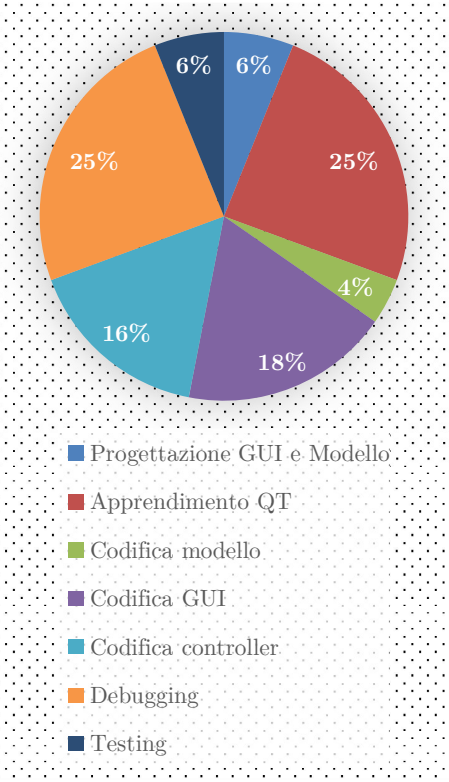
Tutte le informazioni degli item sono salvate in un file XML[2] all'interno della cartella del programma: `./data.xml`. Il file viene caricato ad ogni partenza del programma e salvato ogni volta che si clicca *Save* o *Delete*.



Ad ogni caricamento del programma, oltre al caricamento dei dati tramite il file XML, vengono controllate le scadenze i reminder, sia `Reminder` che `ToDoReminder` (a patto che non siano spuntati). Se un reminder è scaduto viene mostrato una finestra di dialogo che informa della scadenza. Questo controllo avviene anche ogni giorno a mezzanotte.

[2] Si veda *3. Descrizione I/O*

5 Ripartizione ore



- Progettazione GUI e Modello: 3 ore
- Apprendimento QT (tutorato compreso): 12 ore
- Codifica modello: 2 ore
- Codifica GUI: 9 ore
- Codifica controller: 8 ore
- Debugging: 12 ore
- Testing: 3 ore

Totale ore: 49 ore

Nonostante la parte di codifica del modello sia stata veloce la parte relativa alla libreria QT ha preso gran parte del tempo.

6 Suddivisione dei compiti

- Contenitore: Riccardo Pavan
- Controller: Damiano Zanardo – Riccardo Pavan
- Modello
  - `Item`: Damiano Zanardo
  - `ToDo`: Riccardo Pavan
  - `Memo`: Riccardo Pavan
  - `Reminder`: Damiano Zanardo – Riccardo Pavan
  - `ToDoReminder`: Riccardo Pavan
  - `MemoToDo`: Riccardo Pavan
- GUI:
  - `SelectButton`: Riccardo Pavan

- `ItemWidget`: Damiano Zanardo
- `ListItemWidget`: Riccardo Pavan
- Altro: Damiano Zanardo – Riccardo Pavan

7 Ambiente di sviluppo

Ho utilizzato *Visual Studio Code* collegato tramite *ssh* alla macchina virtuale fornita. Tutti i test quindi sono stati effettuati direttamente sulla macchina virtuale.

Per compilare il codice da riga di comando è sufficiente invocare in ordine:

- `qmake QTo-do.pro`
- `make`
- `./QTo-do` per eseguire il programma

File di esempio (`./data.xml`[3]) scaricabile dal seguente link: https://bit.ly/32F68El

[3] Da inserire nella cartella del programma