RELAZIONE PROGRAMMAZIONE AD OGGETTI

QONTAINER

Andrea Favero
Matricola: 1125545

DIPARTIMENTO
MATEMATICA
Tullio Levi Civita

Laurea in Informatica

Anno Accademico 2018/2019

Indice

1 Ambiente di sviluppo

- Sistema operativo:
  - Fedora GNU\Linux 29
  - Ubuntu GNU\Linux 19.04
- Compilatore:
  - Fedora: g++ 8.3.1 20190223 (Red Hat 8.3.1-2)
  - Ubuntu: g++ 8.3.0
- Versione librerie Qt: 5.12.2

2 Ore impiegate:

- analisi preliminare del problema: 2
- progettazione ed implementazione model: 10
- progettazione ed implementazione view: 34
- apprendimento libreria Qt: 1 (perché già imparata gli anni scorsi)
- relazione: 1
- debug + testing: 2
- totali: 50

3 Scopo del progetto

QONTAINER è un'applicazione che permette agli atleti che praticano il nuoto, il ciclismo, la corsa o il triathlon di tenere traccia dei loro allenamenti. Una volta inseriti i dati relativi alle loro prestazioni, essi possono vedere per ciascun allenamento quante calorie hanno bruciato, quanti sali minerali hanno consumato e quanti grammi di peso hanno perso.

QONTAINER permette di inserire, modificare ed eliminare gli *atleti* ed i loro *allenamenti*. Offre inoltre una funzionalità di ricerca che per un dato atleta visualizza i relativi allenamenti (tutti oppure filtrati per sport) che soddisfano certe caratteristiche temporali (una certa data) e prestazionali (*i.e.*: il minimo ed il massimo numero di vasche a dorso che sono state fatte durante un allenamento di nuoto).

Il programma utilizza due contenitori templatizzati, uno per gli atleti di tipo *Contenitore<std::shared_ptr<Persona»*, l'altro per gli allenamenti di tipo *Contenitore<DeepPtr<Allenamento> >*.

4 Descrizione delle gerarchie dei tipi usate

Il progetto presenta tre tipi di gerarchie, una presente nella parte logica e due presenti in quella grafica.

4.1 Gerarchia parte logica

La gerarchia presente nella parte logica è totalmente indipendente dalla libreria *Qt* ed utilizza solamente la libreria standard del linguaggio C++. Tale gerarchia è presente nella figura sottostante.

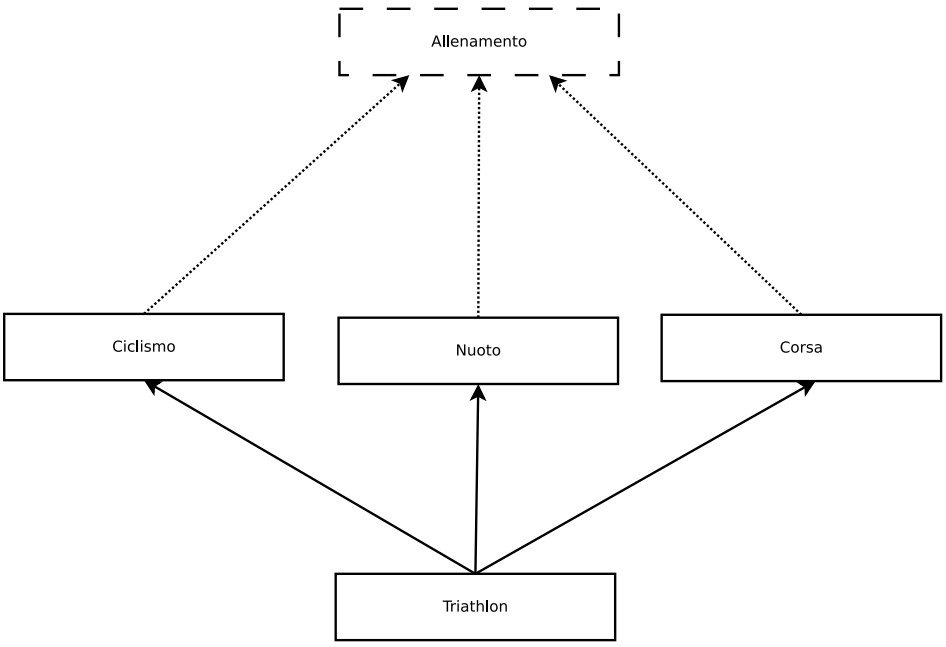

Figura 1: Gerarchia parte logica

La classe base astratta virtuale polimorfa *Allenamento* modella un generico allenamento sportivo portato a termine da un atleta tenendo conto anche della data e della durata in minuti. Il campo dati atleta è di tipo *std::shared_ptr<Persona>* perché tutti gli atleti dell'applicazione sono memorizzati in un contenitore parametrizzato su *std::shared_ptr<Persona>* che è un puntatore smart condiviso, in questo modo, quando un utente modifica un atleta non si crea alcuna discrepanza nei dati dei suoi allenamenti perché rimangono sempre tutti correlati a lui e non ai vecchi nome, cognome e sesso che aveva prima della modifica. Definisce i metodi virtuali:

$\sim$ Allenamento() = default

std::string tipo() const = 0

Allenamento* clone() const = 0

unsigned int calorie() const = 0

unsigned int grassoPerso() const = 0

unsigned int saliMinerali() const = 0

bool operator==(const Allenamento&) const

bool operator>=(const Allenamento&) const

bool operator<=(const Allenamento&) const

Le classi *Nuoto*, *Ciclismo Corsa* e *Triathlon* sono la concretizzazione della classe *Allenamento* da cui derivano. Le classi derivate fanno l'override di tutti i metodi virtuali della classe base ad eccezione del distruttore dato che non ce n'è necessità.

Il metodo **calorie()** permette di sapere quante calorie ha bruciato l'atleta durante l'allenamento, **saliMinerali()** quanti milligrammi di sali minerali sono stati consumati e **grassoPerso()** quanti grammi di grasso sono "andati perduti" per via dell'allenamento.

4.2 Gerarchie parte grafica

La parte grafica è formata da due gerarchie, una (visualizzata nella figura sottostante) riguardante i *delegate* utilizzati per disegnare a video i bottoni contenuti nelle *QTableView* e che permettono di eliminare o modificare un allenamento oppure un atleta, l'altra (nella figura sottostante) riguardante i dialog utilizzati per creare o modificare un certo allenamento.

4.2.1 Gerarchia delegati

La gerarchia dei delegati è composta dalla classe base *DelegatoBottone* che eredita da *QItemDelegate* e fa l'override del distruttore virtuale e dei metodi **paint()** (che si occupa di disegnare a video i bottoni) e **editorEvent()** (che gestisce la pressione del mouse sui bottoni). Le due sottoclassi di *DelegatoBottone* sono *DelegatoEliminazione* e *DelegatoModifica*. La prima è il delegato che gestisce i bottoni di eliminazione (di un atleta o un allenamento) e la seconda è il delegato che gestisce i bottoni di modifica (di un atleta o un allenamento). Nessuna delle due fa l'override dei metodi presenti nella classe base ma, vengono aggiunti semplicemente dei signal e degli slot necessari per eliminare\modificare un allenamento o un atleta.

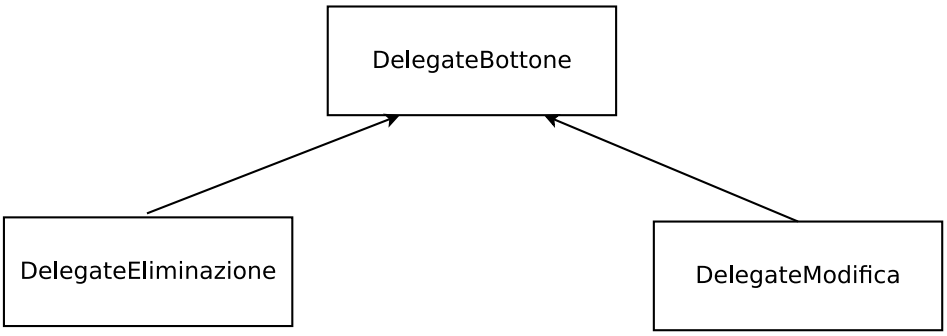

Figura 2: Gerarchia parte logica

4.2.2 Gerarchia dialog

La gerarchia dei *Dialog*, derivante da *QDialog* ha lo scopo di fornire delle finestre di dialogo finalizzate a permettere all'utente di aggiungere o modificare un allenamento. La gerarchia controlla anche che i campi inseriti dall'utente siano validi e, nel caso si accettino le modifiche apportate o che si inserisca un nuovo allenamento, modifica o inserisce l'allenamento nel contenitore e poi avverte il model perché si aggiorni. La classe base *DialogAllenamento* non ha alcun motivo per essere istanziata e per questo definisce i metodi

virtuali puri **setLabelTitolo()**, **inserimentoAllenamento()** e **modificaAllenamento()** (questi ultimi due sono slot).

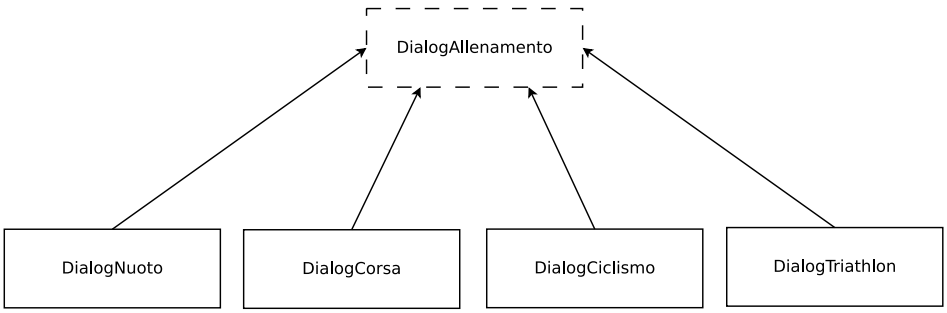

Figura 3: Gerarchia delegati

5 Utilizzo del codice polimorfo

5.1 Codice polimorfo classi allenamenti

I metodi virtuali listati in precedenza vengono richiamati come segue:

- **tipo()**: riga 46 della classe *ModelTabellaAllenamenti*;
- **clone()**: righe 32, 37, 49 della classe *DeepPtr*;
- **calorie()**: riga 53 della classe *ModelTabellaAllenamenti*;
- **grassoPerso()**: riga 55 della classe *ModelTabellaAllenamenti*;
- **saliMinerali()**: riga 57 della classe *ModelTabellaAllenamenti*;
- **operator>=**: righe 34, 36 della classe *SortFilterProxyModelAllenamenti*;
- **operator<=**: righe 34, 36 della classe *SortFilterProxyModelAllenamenti*;
- **operator==()**: riga 175 della classe *DialogTriathlon*, 114 della classe *DialogCiclismo*, 111 della classe *DialogCorsa*, 116 della classe *DialogNuoto*;

5.2 Codice polimorfo classi dialog

Benché le classi della gerarchia reimplementano i metodi virtuali puri della classe base, non viene effettuata alcuna chiamata polimorfa perché tali metodi sono richiamanti solamente all'interno del costruttore della classe. Non c'è alcun motivo per richiamarli al di fuori dato che servono solamente per impostare l'interfaccia grafica dei dialog, cosa che solitamente viene fatta dai costruttori.

6 Manuale utente

6.1 Creazione di un nuovo atleta

La cosa da fare al primo avvio se non si sceglie di utilizzare i file forniti di default è *creare un nuovo atleta*. Bisogna quindi andare sulla tab relativa agli atleti e cliccare sul bottone **Nuovo Atleta** in basso al centro, si aprirà un dialog che permetterà di inserire i dati necessari alla creazione. Una volta creato l'atleta comparirà una nuova riga relativa ad esso sulla tabella e cliccando sui bottoni **modifica** o **elimina** si potrà rispettivamente modificare o eliminare tale atleta.

6.2 Creazione di un nuovo allenamento

Quando nel sistema è presente almeno un atleta è possibile inserire un nuovo allenamento. Per farlo bisogna spostarsi nella tab Allenamenti e cliccare sul bottone dello sport scelto situato in basso al centro. Una volta fatto si aprirà un nuovo dialog che permetterà di istanziare un nuovo allenamento: bisogna selezionare l'atleta desiderato e la data, bisogna immettere la durata in minuti e i milligrammi di sali minerali che sono stati assunti prima o dopo l'allenamento. Se si è scelto il nuoto è necessario che l'atleta abbia nuotato almeno una vasca in uno qualsiasi dei tre stili, se si è scelto il ciclismo oppure la corsa è necessario che abbia percorso almeno un km in una delle qualsiasi specialità disponibili. Per il triathlon che comprende nuoto, corsa e ciclismo sono necessarie le stesse cose. Una volta creato l'allenamento comparirà sulla tabella una nuova riga relativa ad esso in cui, sono presenti dati calcolati in base alle informazioni immesse in fase di creazione. Per rivedere tali informazioni o modificare l'atleta è sufficiente cliccare sul bottone modifica, per eliminarlo basta cliccare su modifica.

7 Ricerca

È possibile effettuare la ricerca sugli allenamenti relativi ad **un** certo atleta. Per farlo bisogna spostarsi nella tab Ricerca (visualizzata nella figura sottostante) e selezionare l'atleta desiderato. Esistono due tipi di ricerca, una generale che filtra tutti gli allenamenti di qualsiasi tipo fatti dall'atleta (bisogna selezionare il radio button **Atleta**) oppure quella relativa ai singoli sport (bisogna selezionare i radio button riguardanti gli sport). Dopodiché è necessario inserire il range di tempo in cui si vogliono cercare gli allenamenti e anche il minimo ed il massimo valore relativo alle caratteristiche proprie degli sport. Per ogni sport è necessario che il valore minimo di **almeno una** sua caratteristica (*i.e.* il numero minimo di vasche a stile, rana o dorso se si prende in considerazione il nuoto) sia ≥ 1. Una volta completato l'inserimento cliccando sul bottone **Cerca** è possibile visualizzare i risultati filtrati a seconda dei dati immessi.

Figura 4: Schermata di ricerca

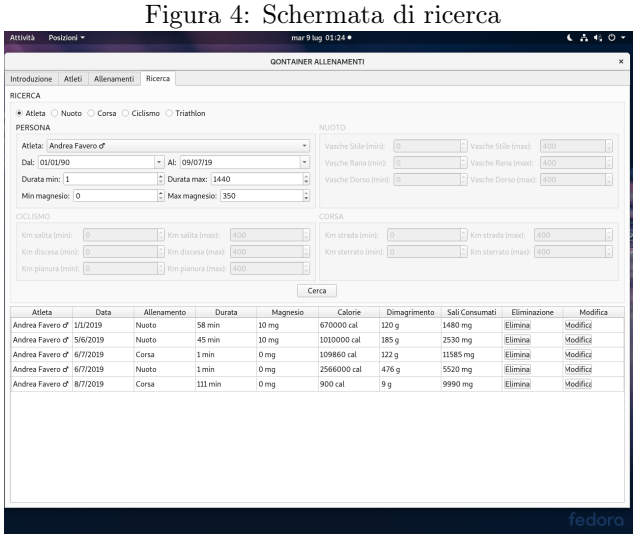

Figura 5: Ricerca

8 Formato file caricamento e salvataggio del contenitore

Il programma utilizza dei file con estensione *.xml* e opera sui file *atleti.xml* ed *allenamenti.xml* che contengono rispettivamente gli atleti che vengono caricati sul contenitore relativo agli atleti e su quello relativo agli allenamenti. Per velocizzare la correzione del progetto al primo avvio viene offerta la possibilità di utilizzare dei file già pronti contenenti atleti ed allenamenti.

9 Indicazioni di compilazione ed esecuzione

La compilazione del progetto necessita del file QONTAINER-MV.pro. Per compilare è necessario dare i comandi:

```
$ qmake
$ make
```

Per avviare l'eseguibile basta dare il comando

```
$ ./QONTAINER-MV
```