



GoFlex

Gazi Adnan Latif

Matricola: 1224442

Progetto di Programmazione ad Oggetti



1 Prefazione

1.1 Abstract

GoFlex è un'applicazione di training semplice, utile e intuitivo da usare. Permette di creare esercizi di attività fisica di diverse tipologie, fornendo specifiche per ognuna, in modo da personalizzare le proprie attività. Gli esercizi vengono poi inseriti in programmi d'allenamento, anch'esse personalizzabili dall'utente. L'applicazione consente di avviare gli allenamenti e fornisce in tempo reale informazioni in base all'andamento dell'allenamento. È anche presente una componente d'intelligenza artificiale che genera informazioni d'interesse all'utente, come ad esempio la stima delle calorie bruciate, la difficoltà delle attività, la durata degli esercizi e altro. Inoltre, l'applicazione è altamente adattabile ad ogni uso in quanto è possibile inserire qualunque tipo di esercizio e creare qualunque tipo di allenamento, quindi perfetta per qualsiasi utente e qualsiasi obiettivo d'allenamento.

1.2 Divisione del tempo

- **Analisi e progettazione:** 9 ore, di cui
 - Analisi dei requisiti: 2 ore
 - Progettazione del diagramma delle classi: 6 ore
 - Progettazione della grafica: 1 ora
- **Realizzazione:** 35 ore, di cui
 - Implementazione del model: 17 ore
 - Implementazione del controller: 4 ore
 - Implementazione della view: 14 ore
- **Apprendimento del framework Qt:** 8 ore
- **Testing e debugging:** 8 ore

Il tempo totale per la realizzazione del progetto è quindi di 60 ore. La soglia massima delle 50 ore imposte dai requisiti del progetto è stata quindi di poco superata. Il motivo è dovuto in parte all'eccessivo tempo richiesto dalla fase di testing e debugging per correggere alcuni problemi di memory leak. Anche la realizzazione del model ha richiesto più tempo del previsto per via della difficoltà nel trovare formule semplici ma al tempo stesso efficaci, da inserire nell'intelligenza artificiale, per l'implementazione di alcune funzionalità del modello. Infine, è stato pensato necessario rafforzare un po' l'interfaccia grafica mediante controlli sui casi limite del programma, per garantire una maggiore usabilità dell'applicazione: quest'operazione ha richiesto un paio d'ore.

1.3 Ambiente di sviluppo

- **Sistema operativo di sviluppo:** Windows 10
- **IDE:** Qt Creator versione 4.13.2
- **Linguaggio di programmazione e framework:** C++ 11 e framework Qt
- **Compilatore:** MinGW 8.1.0

2 Applicazione

2.1 Scelte progettuali

È stato deciso di utilizzare il pattern MVC per lo sviluppo del programma, in quanto le specifiche fornite per la realizzazione del progetto richiedevano esplicitamente una netta separazione, a livello di codice, tra il modello logico e l'interfaccia grafica. Il pattern MVC divide il programma in tre componenti principali: Model, View e Controller. In questo modo ogni parte può essere separata in qualunque momento dalle altre con minimo sforzo, cambiata e infine riconnessa al resto del programma senza necessità di modificare le altre componenti. È stato deciso di implementare prima la parte del modello, in modo da garantire la massima indipendenza del modello logico dalla grafica. Dopodiché è stata sviluppata la grafica, e infine, le due parti sono state collegate tra loro dal controller.

È stato inoltre utilizzato il software GitHub (e un apposito repository Git) per gestire il versionamento del codice sorgente del programma durante l'intero sviluppo del progetto.

2.2 Descrizione dell'applicazione

GoFlex permette all'utente di creare cinque diversi tipi di esercizi, ognuna aventi caratteristiche e obiettivi differenti, di seguito elencate:

- **Cardio:** esercizi cardiovascolari con obiettivo di bruciare grassi.
- **SollevamentoPesi:** esercizi di forza con obiettivo di rafforzare la muscolatura.
- **CrossFit:** esercizi cardiovascolari e di forza con obiettivo di bruciare grassi, rafforzare la muscolatura e migliorare la resistenza fisica. Sono l'unione di esercizi Cardio e SollevamentoPesi.
- **RiposoPassivo:** riposo tra due esercizi in cui si rimane fermi cercando di recuperare.
- **RiposoAttivo:** riposo tra due esercizi in cui si svolge un altro esercizio (Cardio, SollevamentoPesi o CrossFit) per tutto il tempo del riposo, ad un'intensità molto bassa. Esempio: camminare per recuperare da un esercizio di corsa.

Cardio, SollevamentoPesi e CrossFit sono raggruppati nell'entità MonoEsercizio; RiposoPassivo e RiposoAttivo sono raggruppati nell'entità Riposo. MonoEsercizio e Riposo sono a loro volta raggruppati in una generica entità Esercizio. Per ogni esercizio creato bisogna fornire dei relativi dati, in modo da personalizzarli a seconda dell'allenamento che si intende svolgere. Una volta creato un esercizio, questo viene inserito all'interno di una lista d'allenamento: l'utente può navigare tra i vari esercizi, visualizzare statistiche relative al singolo esercizio e all'allenamento complessivo, cambiare o rimuovere esercizi.

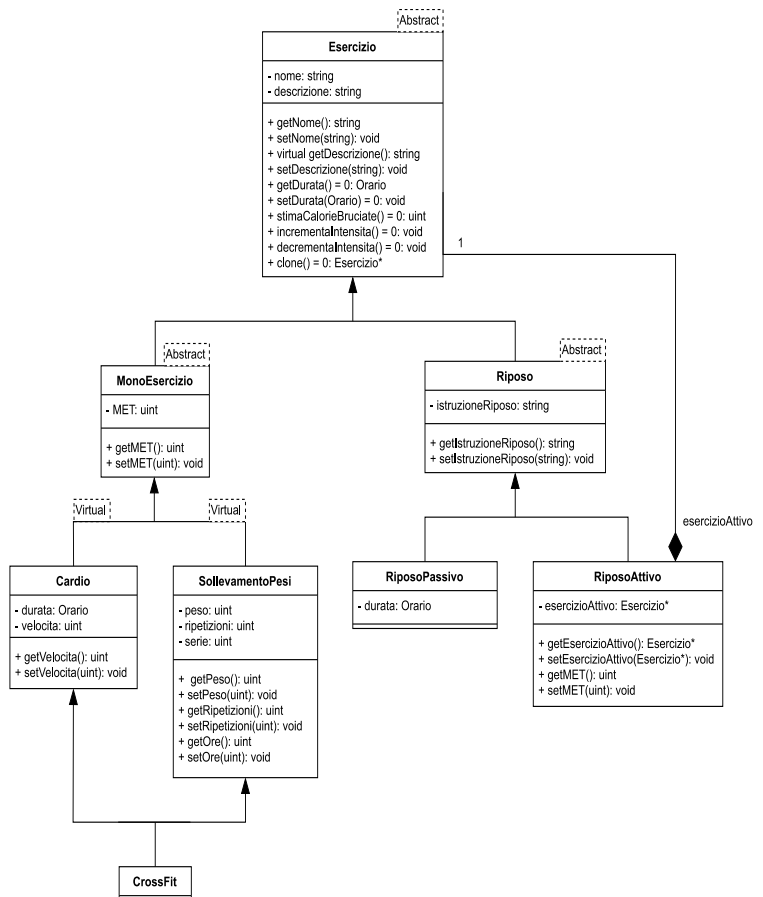
3 Diagramma delle classi

3.1 Model

3.1.1 Gerarchia

I primi due livelli della gerarchia (Esercizio al primo, MonoEsercizio e Riposo al secondo) sono classi astratte per via di metodi virtuali puri e non implementati, pertanto non sono tipi istanziabili. I tipi istanziabili sono le cinque classi agli 'ultimi due livelli. La classe CrossFit, derivando da due classi, Cardio e SollevamentoPesi, impone a quest'ultime due di ereditare virtualmente la loro rispettiva base MonoEsercizio, in modo evitare due copie della base comune MonoEsercizio in CrossFit, e le classiche situazioni di ambiguità dell'ereditarietà a diamante. I metodi virtuali sono presenti nella base della gerarchia e vengono usati in modo polimorfo nel resto del programma. Essi sono:

- **`std::string getDescrizione()`:** restituisce la descrizione fornita dall'utente relativamente all'esercizio. È virtuale perché ogni classe concreta della gerarchia aggiunge automaticamente informazioni su quanto sia faticoso l'esercizio alla descrizione originale.
- **`Orario getDurata()`:** restituisce la durata dell'esercizio nel formato di un oggetto Orario creato appositamente per il



programma. È virtuale perché alcuni esercizi (SollevamentoPesi), in fase di creazione non richiedono la durata dell'esercizio ma lo calcolano alla chiamata di questo metodo a partire dalle altre specifiche dell'esercizio.

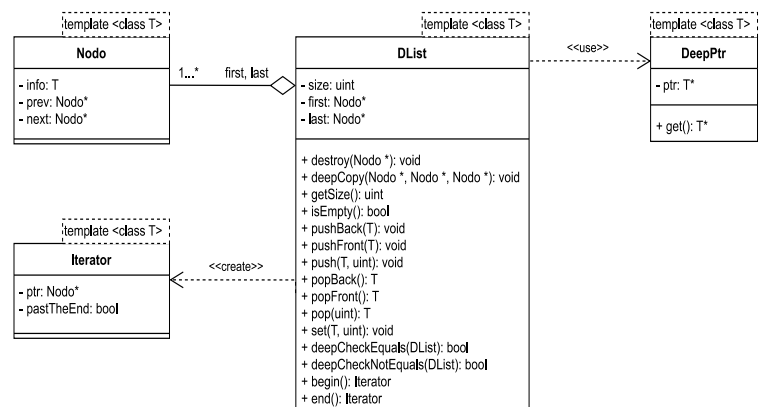
- **`void setDurata(Orario):`** imposta la durata dell'esercizio al valore dell'oggetto Orario passato come parametro. È virtuale perché alcuni esercizi (SollevamentoPesi) calcolano la durata alla chiamata di questo metodo a partire dalle altre specifiche dell'esercizio.
- **`unsigned int stimaCalorieBruciate():`** restituisce un intero positivo riferito alla stima delle calorie che l'esercizio consuma. È virtuale perché il calcolo di questa stima cambia a seconda delle specifiche di ciascun tipo di esercizio.
- **`void incrementalIntensita():`** permette di aumentare l'intensità dell'esercizio, se e finché è possibile. È virtuale perché il modo in cui l'intensità dell'esercizio viene incrementato dipende dalle specifiche di ciascun tipo di esercizio.
- **`void decrementalIntensita():`** permette di diminuire l'intensità dell'esercizio, se e finché è possibile. È virtuale perché il modo in cui l'intensità dell'esercizio viene decrementato dipende dalle specifiche di ciascun tipo di esercizio.
- **`Esercizio * clone():`** restituisce una copia profonda dell'esercizio (profonda grazie al costruttore di copia ridefinito in modo profondo). Il valore restituito è un puntatore polimorfo ad Esercizio, il cui tipo statico è un oggetto Esercizio ma il tipo dinamico è un oggetto concreto della gerarchia. Viene usato in modo polimorfo dagli SmartPointer nel programma.

Vengono quindi rispettati i requisiti del progetto imposti nella gerarchia di almeno tre tipi istanziabili, una classe astratta, un metodo virtuale, una chiamata virtuale e una gerarchia di almeno di due livelli.

3.1.2 Contenitore e SmartPointer

Il contenitore implementato per il progetto si chiama DList ed è una lista doppiamente concatenata: è una classe template, quindi può contenere qualunque tipo di oggetto al suo interno, specificato come parametro template alla creazione del contenitore. Sfrutta la classe Nodo al suo interno per il suo funzionamento e mette a disposizione un iteratore per questa struttura dati attraverso la classe Iterator sempre al suo interno. La classe DList implementa diverse funzioni per la gestione degli elementi contenuti, raggruppabili nel seguente modo: funzioni di aggiunta al

contenitore (in testa, in coda o in una posizione intermedia), funzioni di rimozione dal contenitore (in testa, in coda o in una posizione intermedia), funzioni di sostituzione di elementi (in testa, in coda o in una posizione intermedia). Inoltre, gestisce gli elementi contenuti in modo profondo alla creazione e nella distruzione del contenitore. Implementa anche diversi operatori, tra cui quelli logici, di stampa e di assegnazione per facilitare l'uso della classe. Lo SmartPointer implementato per il progetto si chiama DeepPtr: anch'essa è una classe template, pertanto l'oggetto contenuto può essere qualunque che sia indicato alla creazione dell'oggetto. Rappresenta un puntatore smart e viene usato per contenere puntatori polimorfi della gerarchia, ovvero puntatori ad oggetti Esercizio. Siccome la gerarchia supporta la clonazione e la distruzione profonda, il DeepPtr gestisce in modo profondo gli oggetti puntati. In particolare, ogni volta che si crea un oggetto DeepPtr, questa punta ad una copia dell'oggetto indicato e quando un DeepPtr non viene più utilizzato, viene distrutto anche l'oggetto da esso puntato. Implementa alcuni operatori logici e di dereferenziazione per facilitare il suo uso. Quindi, come da richiesta nelle specifiche del progetto, il DeepPtr, simula gli SmartPointer della libreria standard. Come da richiesta nelle specifiche del progetto, il contenitore contiene oggetti SmartPointer, a loro volta contenenti puntatori polimorfi alla base della gerarchia.



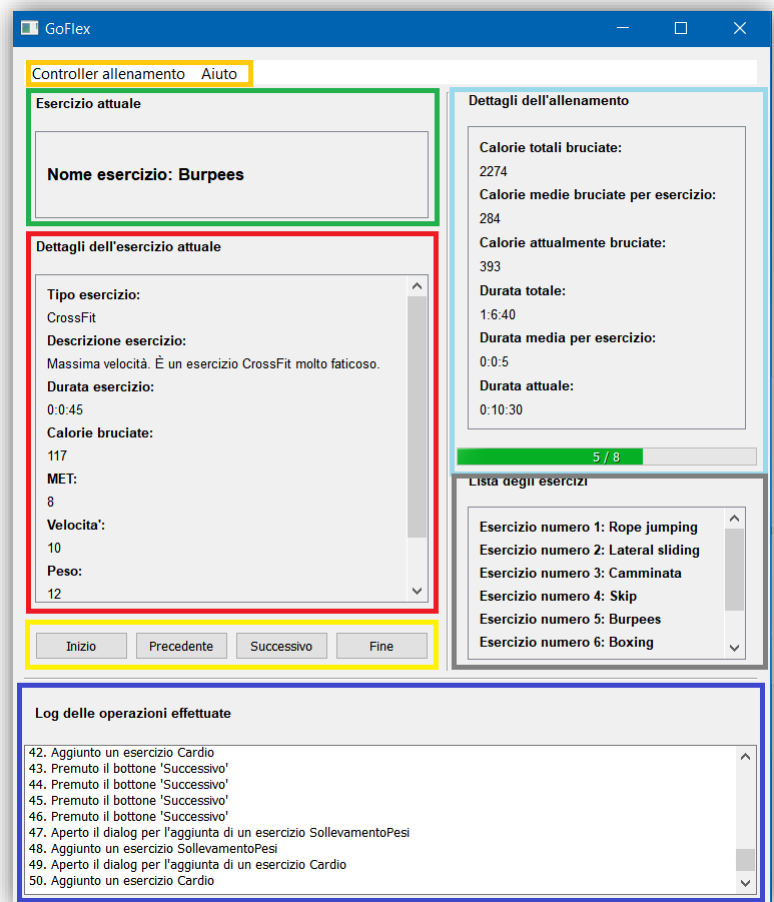
4 Interfaccia grafica

L'interfaccia grafica prevede due principali tipologie di finestre che l'utente può usare per interagire con l'applicazione e dove può visualizzare i propri dati: sono la schermata principale e le diverse schermate dei dialog.

4.1 Schermata principale

La schermata principale è la finestra che viene creata all'avvio dell'applicazione. Inizialmente si presenta priva di contenuto: l'immagine accanto rappresenta la schermata principale catturata in uno stato avanzato d'uso dell'applicazione, in cui erano stati inseriti diversi esercizi. La schermata è suddivisa in sette aree principali, di seguito descritte:

- Menus (in arancione):** è una barra dei menu formata dai tasti controller workout e aiuto. Il primo permette di controllare il proprio allenamento, mentre il secondo, di cercare informazioni utili all'utilizzo dell'applicazione in caso si necessiti di aiuto. Il controller workout consente di aggiungere, cambiare o rimuovere esercizi. In caso si voglia aggiungere o cambiare esercizio, sottomenu permettono di scegliere la tipologia del nuovo esercizio, e in caso di esercizi composti da altri esercizi (RiposoAttivo) di scegliere il tipo dell'esercizio che la compone. L'aiuto permette di visualizzare finestre di dialogo esplicative di cos'è il MET, il CrossFit e il RiposoAttivo.
- Esercizio attuale (in verde):** mostra il nome dell'esercizio attuale in grande cosicché sia ben visibile.
- Dettagli dell'esercizio attuale (in rosso):** mostra tutti i dettagli dell'esercizio attuale. Vengono anche inserite informazioni non indicate dall'utente alla creazione dell'esercizio, ma bensì ricavate dalla componente di intelligenza artificiale del programma (il tipo di esercizio, informazioni sulla faticosità dell'esercizio aggiunto alla descrizione originale, la durata nel caso di esercizi SollevamentoPesi e la stima delle calorie bruciate).
- Controller degli esercizi (giallo):** è un'area per la navigazione tra esercizi. È formata da quattro bottoni che permettono rispettivamente di andare al primo esercizio dell'allenamento, all'esercizio precedente, all'esercizio successivo, all'ultimo esercizio dell'allenamento. Nel caso in cui non ci dovessero essere esercizi nell'allenamento, l'interazione con questi bottoni non provocano alcun'azione.
- Dettagli dell'allenamento (in azzurro):** riporta informazioni statistiche relative all'allenamento, quali le calorie totali/medie/attuali bruciate, la durata totale/media/attuale degli esercizi. Quest'area comprende anche un progress bar che mostra lo stato di avanzamento dell'allenamento, indicando il numero totale di esercizi nell'allenamento e il numero dell'esercizio corrente.
- Lista degli esercizi (in grigio):** visualizza gli esercizi che compongono l'allenamento, nel loro ordine di inserimento.



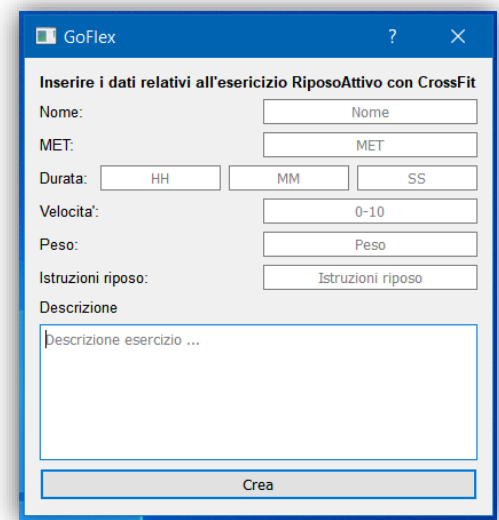
- **Sessione di log (in blu):** stampa informazioni relative a tutte le interazioni effettuate dall'utente col programma durante l'utilizzo dell'applicazione. Ogni operazione è numerata e ordinata, cosicché le si possano ripercorrere all'indietro le azioni in caso di necessità.

4.2 Dialog

I Dialog sono finestre di pop-up che vengono generate a partire dalla schermata principale quando vengono eseguite operazioni specifiche dall'utente. Possono essere per la creazione di esercizi, per l'inserimento numerico o per la visualizzazione di vari messaggi. Questi sono widget creati su misura appositamente per il programma.

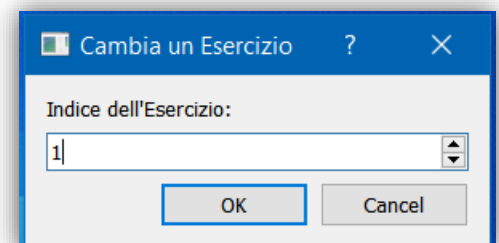
4.2.1 Dialog per la creazione di esercizi

Questi dialog vengono generati ogni volta che bisogna aggiungere o cambiare un esercizio. A seconda del tipo dell'esercizio che si vuole creare, viene generato un dialog differente, uguale nella struttura della finestra, ma diverso nel contenuto: ciò rende molto modulare l'utilizzo dei dialog. Questi dialog contengono campi in cui si possono inserire le specifiche dell'esercizio che si vuole creare. Le informazioni inserite vengono poi acquisite e sincronizzate nella schermata principale in modo da poter visualizzare l'esercizio creato. Si noti che queste finestre godono di una grande flessibilità nell'inserimento dei dati: valori non inseriti o inseriti in formato non corretto vengono convertiti a valori di default in base ai criteri della creazione dell'esercizio.



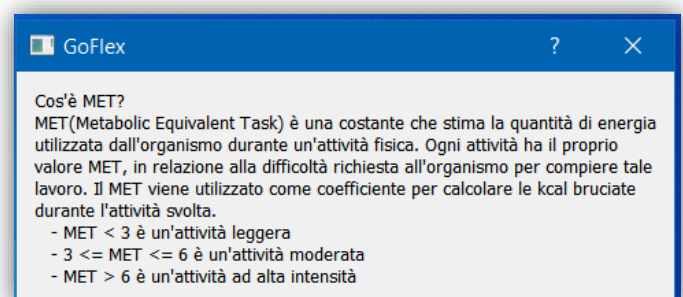
4.2.2 Dialog per l'inserimento numerico

Questi dialog vengono generati ogni volta che l'utente vuole cambiare o rimuovere un esercizio e quindi si necessita di sapere con un valore numerico la posizione in cui effettuare tale operazione. Questi dialog permettono di inserire solo valori interi e positivi maggiori di zero (valore 1 indica il primo esercizio della lista, ovvero quello in posizione 0). Inoltre, non è possibile richiedere tale dialog in caso in cui la lista degli esercizi sia vuota, in quanto non è possibile cambiare o rimuovere alcun elemento. Non è nemmeno possibile inserire valori illeciti: ogni utilizzo illecito di questo tipo di dialog genera una finestra che comunica l'errore commesso. Nel caso in cui si voglia cambiare un esercizio, viene prima generato questo tipo di dialog, seguito poi dal classico dialog per la creazione dell'esercizio.



4.2.3 Dialog per la visualizzazione di messaggi

Questi dialog vengono generati ogni volta che bisogna comunicare qualche messaggio all'utente, quindi per messaggi di errori nell'utilizzo dell'applicazione o quando l'utente chiede aiuto nella sezione dedicata. Si consiglia in particolare di leggere il messaggio generato in 'Aiuto -> Cos'è MET?' perché il MET è un valore numerico richiesto alla creazione di ogni esercizio e da esso dipendono molte funzionalità del programma.



5 Note aggiuntive

5.1 Gestione degli errori

L'applicazione gestisce gli errori e i casi limiti d'uso a due livelli: frontend e backend. A livello frontend vengono effettuati test nel controller, generando dialog contenenti messaggi d'errore. Mentre a livello backend vengono effettuati controlli nelle classi della gerarchia, del contenitore, nelle classi di gestione del model e nelle classi ausiliarie, dove vengono utilizzati i tipi di eccezioni messe a disposizione dalla libreria standard `std::runtime_error` e `std::logic_error` per segnalare errori causati da eventi non predicibili durante la compilazione ed errori logici nell'uso del programma. L'intento dell'applicazione è di gestire gli errori in modo da non causare l'arresto dell'applicazione: in questo modo si riesce a garantire la correttezza dell'interazione dell'utente con l'applicazione e del giusto funzionamento del programma. L'applicazione è quindi molto robusta e al tempo stesso flessibile: permette all'utente di fare operazioni critiche, come ad esempio inserire lettere al posto di valori numerici relativi ai dati degli esercizi, senza causare l'arresto dell'applicazione.

5.2 Evolvibilità futura dell'applicazione

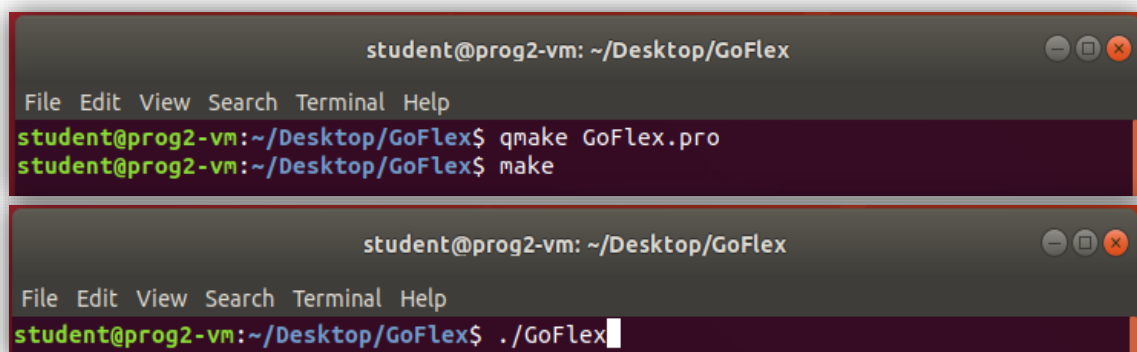
L'abstract del progetto e la modularità del codice garantiscono una buona evolvibilità futura dell'applicazione: il progetto, infatti, può essere ripreso in futuro e implementato aggiungendo componenti senza dover modificare le parti già presenti. Un esempio di evolvibilità futura del programma è la seguente: aggiunta della sezione dedicata agli allenamenti e la loro relativa gestione. L'entità Allenamento diverrebbe una base gerarchica da cui derivare le classi concrete AllenamentoSingolo e RoutineAllenamento, riferiti rispettivamente ad un allenamento occasionale e ad una scheda di allenamento. Si noti che i metodi `Esercizio::incrementaIntensita()` e `Esercizio::decrementaIntensita()` non vengono utilizzati in quanto l'interfaccia grafica non permette di raggiungere tali comandi: in futuro si potrebbe sviluppare una sezione apposita nella grafica. Inoltre, un timer automatico, salvataggio e condivisione dei dati sono solo alcune idee di evolvibilità futura.

6 Materiale consegnato e compilazione

Viene consegnata la cartella *GoFlex.zip*, in cui sono presenti i seguenti file:

- Il file *GoFlex.pro* del programma
- Tutti i file *.h* del programma
- Tutti i file *.cpp* del programma
- Il file *GoFlex.ico* per l'icona dell'applicazione
- Il file *GoFlex.pdf* per la relazione del progetto

Il comando `qmake -project` non permette di generare correttamente il file *.pro* del programma e pertanto, non consente una corretta compilazione dell'applicazione mediante il comando `make`. È quindi necessario il file *.pro* originale del programma, che quindi viene allegato nella cartella consegnata. Estratto la cartella in formato *.zip* e posizionatosi col terminale al suo interno, bisogna eseguire i seguenti comandi nel seguente ordine per avviare l'applicazione: `qmake GoFlex.pro`, `make`, `./GoFlex`.



```
student@prog2-vm: ~/Desktop/GoFlex
File Edit View Search Terminal Help
student@prog2-vm:~/Desktop/GoFlex$ qmake GoFlex.pro
student@prog2-vm:~/Desktop/GoFlex$ make

student@prog2-vm: ~/Desktop/GoFlex
File Edit View Search Terminal Help
student@prog2-vm:~/Desktop/GoFlex$ ./GoFlex
```