Oggetto: Relazione progetto Programmazione a oggetti
Studente: Vedovato Alex, mat.2042353

Titolo: Gestore di allenamenti

Introduzione

GestoreAllenamenti è un'applicazione desktop che permette all'utente di creare e gestire diversi piani di allenamento attraverso l'inserimento, la modifica, la cancellazione e la visione dei dettagli di esercizi di vario tipo. Quest'ultimi, infatti, vengono differenziati in base alle loro caratteristiche e modalità di esecuzione, dando luogo a visualizzazioni grafiche diverse a seconda del fatto che si tratti di un esercizio a tempo, a ripetizioni, a serie e ripetizioni oppure a serie e tempo.

L'utente attraverso l'utilizzo di GestoreAllenamenti, una volta deciso se creare un nuovo allenamento oppure caricarne uno già esistente, ha la piena libertà di poter aggiungere al suo piano ciò che preferisce da una lista di esercizi disponibili (la quale può anche essere filtrata in diversi modi, per esempio per gruppo muscolare allenato), potendo anche modificare subito, oppure in seguito, le caratteristiche modificabili di un esercizio, come nome o descrizione, ma soprattutto quegli attributi che vanno a variare la modalità di esecuzione dell'esercizio, come numero di ripetizioni o durata. Infine, come già detto, ogni esercizio può essere rimosso dall'allenamento, oppure se ne possono vedere i dettagli (diversi a seconda del tipo), dato che nella schermata principale si vedono solo le informazioni più rilevanti.

Ho scelto questo tema poiché la palestra e l'allenamento sono una mia passione, ed inoltre ho pensato che le diverse funzionalità implementate e la struttura delle classi potessero essere ottime per rispettare i vincoli imposti dal progetto.

Descrizione del modello

All'interno del modello logico, come si può anche vedere dal diagramma UML riportato nell'ultima pagina della relazione, si trovano la gestione delle diverse tipologie di esercizi, la parte relativa all'allenamento, che non è altro che un contenitore di esercizi, ed infine una classe per la gestione della persistenza dei dati.

Il modello parte da una classe astratta *Esercizio*, la quale rappresenta le informazioni comuni a tutti gli esercizi possibili. Qui ho subito dovuto fare due scelte importanti, ovvero decidere come gestire l'informazione riguardante i gruppi muscolari allenati da un esercizio, e quali attributi rendere modificabili o meno. Per quanto riguarda la prima scelta, la mia decisione è stata quella di utilizzare un'enumerazione *GruppoMuscolare* per rappresentare i vari gruppi muscolari, e di sfruttare un array dinamico di elementi di quest'ultima; questo perché il numero di gruppi muscolari allenati da un esercizio è variabile, ma comunque limitato e non destinato a cambiare nel tempo, e, dunque, nel mio caso un array dinamico è preferibile ad una struttura dati più complessa. Invece, quanto a cosa rendere modificabile o meno, ho scelto di non rendere modificabili i gruppi muscolari allenati da un esercizio e il fatto che esso sia cardio oppure non lo sia; questo perché non ritengo sensato che vengano modificate queste informazioni all'interno di un esercizio, poiché lo cambierebbero completamente, facendolo diventare un altro esercizio totalmente diverso. Mentre, per quanto riguarda i campi dati che non cambiano il senso logico di un esercizio, ovvero nome e descrizione, questi sono dotati anche di metodi setter, oltre che di metodi getter. Lo stesso ragionamento logico viene applicato anche ai campi dati delle sottoclassi

di *Esercizio*, per i quali si può vedere se è presente un metodo setter o meno all'interno del diagramma UML.

Da *Esercizio* derivano virtualmente poi 3 classi:

-*EsercizioTempo*: classe concreta che rappresenta la categoria di esercizi la cui esecuzione prevede lo svolgimento di una determinata attività per un certo periodo di tempo;

-*EsercizioRipetizioni*: classe concreta che rappresenta la categoria di esercizi la cui esecuzione prevede lo svolgimento di una determinata attività per un certo numero di ripetizioni;

-*EsercizioSerie*: classe astratta che rappresenta la categoria di esercizi la cui esecuzione prevede lo svolgimento di una determinata attività per più serie, intervallate da un periodo di riposo;

Inoltre, con eredità a diamante, troviamo le ultime due categorie di esercizi: *EsercizioSerieTempo* e *EsercizioSerieRipetizioni*. Queste due classi rappresentano concretamente quegli esercizi la cui esecuzione prevede il compimento per più serie di una determinata attività, andando a definire il tipo di svolgimento di quest'ultima (a tempo o a ripetizioni).

Tutti gli esercizi hanno una tipologia di esecuzione e un numero di calorie bruciate, i quali sono calcolati diversamente a seconda del tipo concreto dell'esercizio e sono basati sui valori degli attributi dell'esercizio in questione. Per esempio, il calcolo delle calorie in *EsercizioRipetizioni* avviene moltiplicando il numero di ripetizioni per le calorie bruciate da ogni ripetizione, mentre in *EsercizioTempo* si moltiplica la durata dell'esercizio in minuti per le calorie bruciate per minuto di esecuzione dell'esercizio.

Successivamente, per consentire l'arricchimento in maniera dinamica delle classi che compongono la gerarchia ho scelto di utilizzare il design pattern Visitor. A tale scopo è stata dunque realizzata la classe astratta EsercizioVisitorInterface e, di conseguenza, in *Esercizio* è stato inserito il metodo virtuale puro accept, ma di questo se ne parla meglio nella sezione dedicata al polimorfismo.

Passando ora alla parte relativa alla classe *Allenamento*, ho scelto di utilizzare come modello di contenitore per i diversi esercizi Vector, adattandolo alle mie necessità. Questa decisione è stata presa prevalentemente per questioni di efficienza, difatti il principale uso dell'applicazione che prevedo da parte dell'utente è quello di consultare gli esercizi di un allenamento, visualizzandone i dettagli, mentre la creazione di nuovi allenamenti o la modifica di quelli già esistenti saranno sicuramente operazioni importanti, ma in ogni caso secondarie. Dunque ciò che mi occorreva era un contenitore che mi garantisse un accesso rapido ai suoi elementi, indipendentemente dalla loro posizione, per questo ho scelto di implementare nativamente una mia versione di Vector, la quale supporta l'accesso casuale agli elementi in tempo costante. Inoltre, anche volendo preoccuparsi degli inserimenti o delle cancellazioni di esercizi all'interno dell'allenamento, solitamente quando si pianifica un allenamento si procede in ordine, per cui eventuali operazioni di questi tipi avverrebbero quasi sempre in coda, e quindi con un tempo ammortizzato costante per quanto riguarda questo contenitore.

Infine, per quanto riguarda la classe GestoreCSV, questa è presente per occuparsi dalla persistenza dei dati, creando, aggiornando o leggendo file CSV, e sfrutta la classe QTextStream, un ottimo strumento offerto da Qt per svolgere queste operazioni di I/O.

Polimorfismo

L'utilizzo principale del polimorfismo non banale riguarda il design pattern Visitor nella gerarchia *Esercizio*. Esso viene utilizzato per la costruzione di widget differenti a seconda del tipo concreto dell'esercizio, difatti la visualizzazione di un esercizio prevede quattro possibili rese grafiche, una per ciascuna delle quattro categorie concrete di esercizi. Nel dettaglio si ha che se l'esercizio è di tipo *EsercizioTempo* allora nella visualizzazione di quest'ultimo, all'interno di uno qualsiasi dei dialog dove è possibile visualizzare un esercizio, si avrà la durata suddivisa in ore, minuti e secondi, mentre per quanto riguarda *EsercizioRipetizioni* ciò che si vedrà sarà il numero di ripetizioni dell'esercizio. Infine se l'esercizio è di tipo *EsercizioSerieTempo* o *EsercizioSerieRipetizioni*, oltre all'informazione relativa alla singola serie (durata o numero di ripetizioni), si visualizzeranno anche il numero di serie e il recupero tra una serie e l'altra suddiviso in minuti e secondi.

Per la costruzione di questi diversi widget è stata implementata *EsercizioVisitorInteface*, da cui deriva poi la classe *EsercizioInfoVisitor*, la quale attraverso i suoi metodi restituisce elementi grafici diversi in base al tipo concreto dell'oggetto visitato. Da notare che nel metodo accept, per evitare la duplicazione del codice, attraverso una variabile booleana *readonly* segnalo al Visitor se i campi dati dell'esercizio dovranno essere modificabili o meno.

Persistenza dei dati

Per la persistenza dei dati viene utilizzato il formato CSV. Si ha dunque un unico file per allenamento, contenente su ogni riga il valore dei campi dati di un esercizio, separati dal carattere ';'. Per riuscire a riconoscere la classe dell'esercizio da ricostruire una volta che viene letto il file, prima del valore degli attributi dell'oggetto si trova una stringa che ne rappresenta il tipo, come ad esempio "EsercizioTempo". La gestione della persistenza dei dati con questo formato avviene all'interno della classe GestoreCSV, dove per la scrittura del file si richiama il metodo virtuale to_CSV() degli esercizi che compongono l'allenamento, mentre per la lettura si richiama il giusto costruttore in base al valore della "stringa tipo".

Un esempio della struttura dei file è dato dal file "AllenamentoEsempio.csv" fornito assieme al codice all'interno della cartella "I tuoi allenamenti", in cui troviamo un allenamento contenente esercizi di tutti i tipi, in modo da poterne visualizzare la diversa struttura e testare il comportamento dell'applicazione con ognuno di questi.

Funzionalità implementate

Le funzionalità implementate sono, per semplicità, suddivise in due categorie: funzionali ed estetiche. Le prime comprendono:

- gestione di quattro tipologie di esercizi concreti
- gestione di due problemi di ereditarietà a diamante
- gestione della memoria profonda, dovuta all'utilizzo di un array dinamico all'interno della classe *Esercizio*
- implementazione di un template per la ricerca in base alla classe di esercizi di un certo tipo (o derivati) all'interno di un allenamento
- implementazione di *iterator* e *const_iterator* per la classe/il contenitore *Allenamento*

Le funzionalità grafiche:

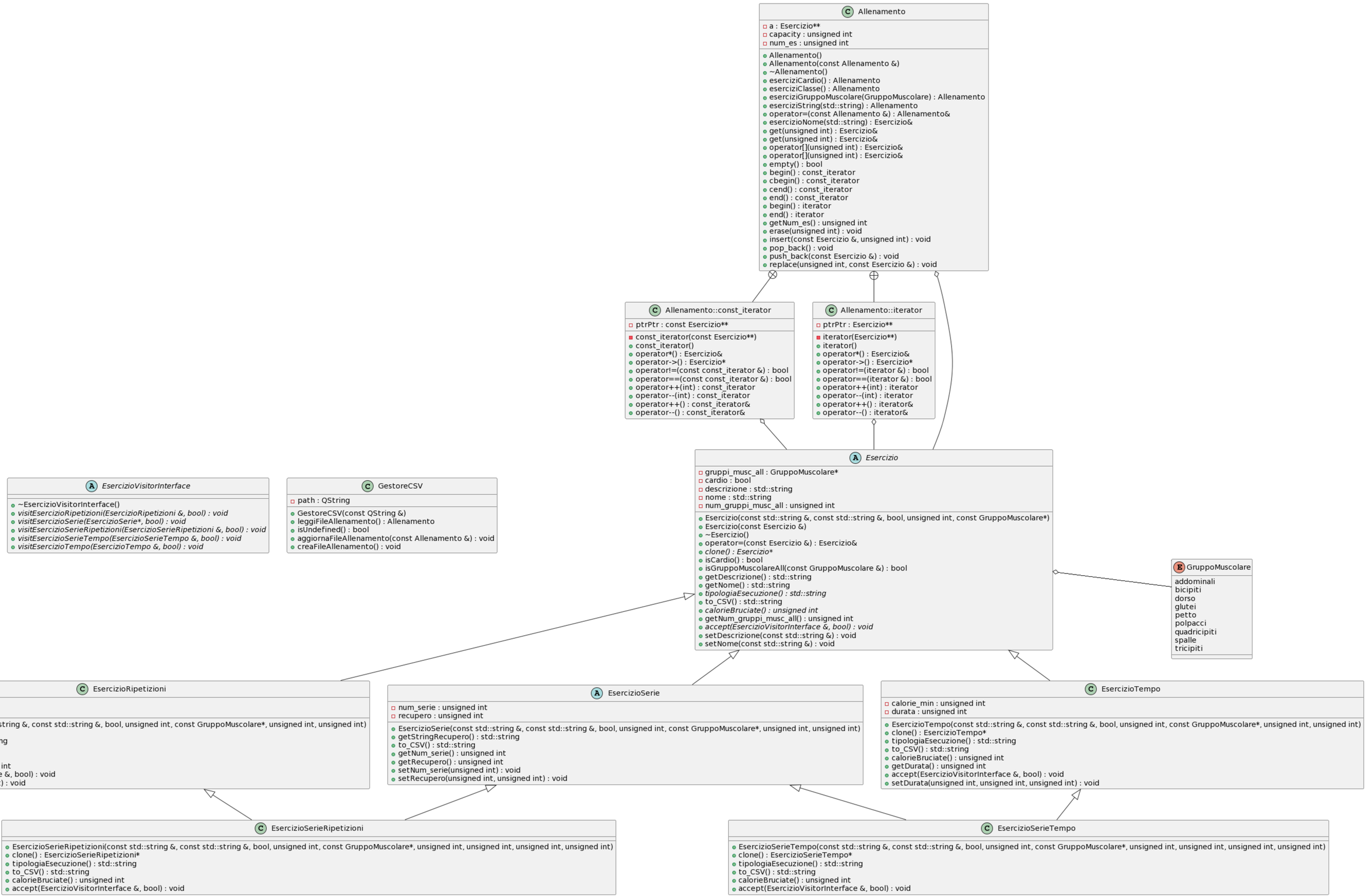
- barra dei menù in alto
- utilizzo di una toolbar attivabile/disattivabile (tramite menù)
- utilizzo di icone nella toolbar e nelle voci del menù
- status bar in fondo alla finestra
- scorciatoie da tastiera (mostrate anche nelle voci del menù)
- controllo della presenza di modifiche non salvate prima di uscire
- controllo della presenza di modifiche non salvate prima dell'apertura di un allenamento
- gestione del ridimensionamento
- ogni tipologia di esercizio ha una propria visualizzazione
- utilizzo di icone nei pulsanti
- utilizzo di immagini nella visualizzazione degli esercizi
- selezione degli esercizi dalla tabella nella schermata principale con singolo click del mouse e deselezione attraverso doppio click
- possibilità di filtrare gli esercizi per tipo o per gruppo muscolare allenato all'interno del dialog di aggiunta di un esercizio
- segnalazione di eventuali problemi all'utente attraverso l'uso di QMessageBox (e.g. input mancante)

Le funzionalità elencate sono intese in aggiunta a quanto richiesto dalle specifiche del progetto.

Rendicontazione ore

Attività	Ore Previste	Ore Effettive
Studio e progettazione	10	11
Sviluppo del codice del modello	10	9
Studio del framework Qt	10	11
Sviluppo del codice della GUI	10	14
Test e debug	5	5
Stesura della relazione	5	5
totale	50	**55**

Il monte ore è stato leggermente superato in quanto studio del framework QT e sviluppo del codice della GUI hanno richiesto più tempo di quanto previsto, in particolare, vista la mia inesperienza per quanto riguarda il framework QT, la ricerca e il corretto utilizzo di widget che calzassero a pennello con la mia applicazione mi ha richiesto uno sforzo maggiore rispetto a quanto pianificato all'inizio.


Allenamento
- a : Esercizio**
- capacity : unsigned int
- num_es : unsigned int
+ Allenamento()
+ Allenamento(const Allenamento &)
+ ~Allenamento()
+ eserciziCardio() : Allenamento
+ eserciziClasse() : Allenamento
+ eserciziGruppoMuscolare(GruppoMuscolare) : Allenamento
+ eserciziString(std::string) : Allenamento
+ operator=(const Allenamento &) : Allenamento&
+ esercizioNome(std::string) : Esercizio&
+ get(unsigned int) : Esercizio&
+ get(unsigned int) : Esercizio&
+ operator[](unsigned int) : Esercizio&
+ operator[](unsigned int) : Esercizio&
+ empty() : bool
+ begin() : const_iterator
+ cbegin() : const_iterator
+ cend() : const_iterator
+ end() : const_iterator
+ begin() : iterator
+ end() : iterator
+ getNum_es() : unsigned int
+ erase(unsigned int) : void
+ insert(const Esercizio &, unsigned int) : void
+ pop_back() : void
+ push_back(const Esercizio &) : void
+ replace(unsigned int, const Esercizio &) : void
Allenamento::const_iterator
- ptrPtr : const Esercizio**
- const_iterator(const Esercizio**)
+ const_iterator()
+ operator*() : Esercizio&
+ operator->() : Esercizio*
+ operator!=(const const_iterator &) : bool
+ operator==(const const_iterator &) : bool
+ operator++(int) : const_iterator
+ operator--(int) : const_iterator
+ operator++() : const_iterator&
+ operator--() : const_iterator&
Allenamento::iterator
- ptrPtr : Esercizio**
- iterator(Esercizio**)
+ iterator()
+ operator*() : Esercizio&
+ operator->() : Esercizio*
+ operator!=(iterator &) : bool
+ operator==(iterator &) : bool
+ operator++(int) : iterator
+ operator--(int) : iterator
+ operator++() : iterator&
+ operator--() : iterator&
Esercizio
- gruppi_musc_all : GruppoMuscolare*
- cardio : bool
- descrizione : std::string
- nome : std::string
- num_gruppi_musc_all : unsigned int
+ Esercizio(const std::string &, const std::string &, bool, unsigned int, const GruppoMuscolare*)
+ Esercizio(const Esercizio &)
+ ~Esercizio()
+ operator=(const Esercizio &) : Esercizio&
+ clone() : Esercizio*
+ isCardio() : bool
+ isGruppoMuscolareAll(const GruppoMuscolare &) : bool
+ getDescrizione() : std::string
+ getNome() : std::string
+ tipologiaEsecuzione() : std::string
+ to_CSV() : std::string
+ calorieBruciate() : unsigned int
+ getNum_gruppi_musc_all() : unsigned int
+ accept(EsercizioVisitorInterface &, bool) : void
+ setDescrizione(const std::string &) : void
+ setNome(const std::string &) : void
GruppoMuscolare
addominali
bicipiti
dorso
glutei
petto
polpacci
quadricipiti
spalle
tricipiti
EsercizioVisitorInterface
+ ~EsercizioVisitorInterface()
+ visitEsercizioRipetizioni(EsercizioRipetizioni &, bool) : void
+ visitEsercizioSerie(EsercizioSerie*, bool) : void
+ visitEsercizioSerieRipetizioni(EsercizioSerieRipetizioni &, bool) : void
+ visitEsercizioSerieTempo(EsercizioSerieTempo &, bool) : void
+ visitEsercizioTempo(EsercizioTempo &, bool) : void
GestoreCSV
- path : QString
+ GestoreCSV(const QString &)
+ leggiFileAllenamento() : Allenamento
+ isUndefined() : bool
+ aggiornaFileAllenamento(const Allenamento &) : void
+ creaFileAllenamento() : void
EsercizioRipetizioni
- calorie_rep : unsigned int
- num_ripetizioni : unsigned int
+ EsercizioRipetizioni(const std::string &, const std::string &, bool, unsigned int, const GruppoMuscolare*, unsigned int, unsigned int)
+ clone() : EsercizioRipetizioni*
+ tipologiaEsecuzione() : std::string
+ to_CSV() : std::string
+ calorieBruciate() : unsigned int
+ getNum_ripetizioni() : unsigned int
+ accept(EsercizioVisitorInterface &, bool) : void
+ setNum_ripetizioni(unsigned int) : void
EsercizioSerie
- num_serie : unsigned int
- recupero : unsigned int
+ EsercizioSerie(const std::string &, const std::string &, bool, unsigned int, const GruppoMuscolare*, unsigned int, unsigned int)
+ getStringRecupero() : std::string
+ to_CSV() : std::string
+ getNum_serie() : unsigned int
+ getRecupero() : unsigned int
+ setNum_serie(unsigned int) : void
+ setRecupero(unsigned int, unsigned int) : void
EsercizioTempo
- calorie_min : unsigned int
- durata : unsigned int
+ EsercizioTempo(const std::string &, const std::string &, bool, unsigned int, const GruppoMuscolare*, unsigned int, unsigned int)
+ clone() : EsercizioTempo*
+ tipologiaEsecuzione() : std::string
+ to_CSV() : std::string
+ calorieBruciate() : unsigned int
+ getDurata() : unsigned int
+ accept(EsercizioVisitorInterface &, bool) : void
+ setDurata(unsigned int, unsigned int, unsigned int) : void
EsercizioSerieRipetizioni
+ EsercizioSerieRipetizioni(const std::string &, const std::string &, bool, unsigned int, const GruppoMuscolare*, unsigned int, unsigned int, unsigned int, unsigned int)
+ clone() : EsercizioSerieRipetizioni*
+ tipologiaEsecuzione() : std::string
+ to_CSV() : std::string
+ calorieBruciate() : unsigned int
+ accept(EsercizioVisitorInterface &, bool) : void
EsercizioSerieTempo
+ EsercizioSerieTempo(const std::string &, const std::string &, bool, unsigned int, const GruppoMuscolare*, unsigned int, unsigned int, unsigned int, unsigned int)
+ clone() : EsercizioSerieTempo*
+ tipologiaEsecuzione() : std::string
+ to_CSV() : std::string
+ calorieBruciate() : unsigned int
+ accept(EsercizioVisitorInterface &, bool) : void