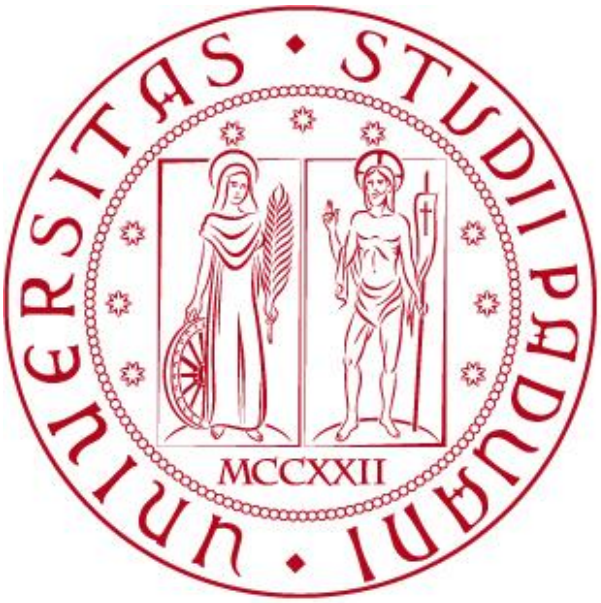


UNIVERSITÀ DEGLI STUDI DI PADOVA

Progetto di Programmazione ad Oggetti

Anno accademico: 2021/2022

Studente: Cusin Matteo

Matricola: 2008073

Data: 26/06/2022

TT-Training Tracker

Introduzione

Il progetto sviluppato è un programma relativo alla gestione di un **piano di allenamenti**, compilabile e modificabile nel tempo, dal quale estrarre, per mezzo di grafici, delle informazioni utili per la comprensione dello stato di forma dell'atleta.

La possibilità di modifica del piano non è limitata nel tempo: si possono "impostare" allenamenti anche in un tempo futuro rispetto a quello di uso dell'applicazione.

Gli allenamenti devono sottostare ad alcuni **vincoli** logici:

- Avere **durata** minore di 20 ore e non nulla;
- Non avere **sovrapposizioni temporali** con altri allenamenti;
- Avere **nome** non nullo;
- Avere un **numero di esercizi** compreso tra 1 e 15 (allenamenti di "ripetizione", descritti nella sezione "Gerarchie di tipi");
- Avere una "**distanza percorsa**" non nulla (allenamenti di "resistenza", descritti nella sezione "Gerarchie di tipi").

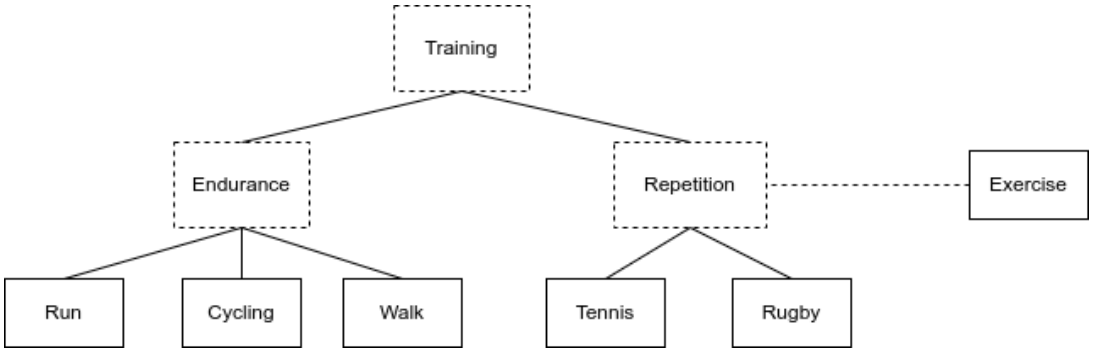
Anche gli esercizi, inclusi in alcuni tipi di allenamenti (come già anticipato), devono sottostare ad alcuni vincoli:

- Avere **nome** non nullo;
- Avere **tempi** di durata e recupero non nulli.

Gerarchie di tipi

E' stato utilizzato il pattern architetturale "**Model-View-Presenter**" (derivazione di "Model-View-Controller") in modo da garantire massima separazione tra vista e modello, delegando al controller l'onere di traduzione dei dati tra la vista ed il modello.

Il progetto TT-Training Tracker prevede **tre gerarchie** distinte; la prima, appartenente al modello, si compone delle seguenti classi:



La classe “**Training**” è la classe base astratta della gerarchia: essa gestisce le informazioni comuni a tutti gli allenamenti, ovvero il nome e la data-ora di inizio.

La classe “**Endurance**” è classe astratta che eredita da “Training” e gestisce le informazioni per gli allenamenti di resistenza, ovvero la distanza percorsa (durante l’allenamento) ed il tempo impiegato; tale classe implementa il metodo virtuale puro “getDuration()” della classe “Training”.

La classe “**Exercise**” modella un esercizio avente nome, durata dell’attività e durata del recupero.

La classe “**Repetition**” è classe astratta che eredita da “Training” e che gestisce una collezione di esercizi (si fa uso di un **std::vector**) per gli allenamenti di ripetizione; la scelta del “std::vector” è fatta in considerazione della quantità ridotta (minimo 1, massimo 15) di esercizi assegnabili ad ogni allenamento.

La classe “Repetition”, inoltre, implementa il metodo virtuale puro “getDuration()” della classe “Training” sommando le durate di attività e di recupero di tutti gli esercizi; implementa anche il distruttore virtuale (sempre di “Training”) consentendo la deallocazione di tutti gli esercizi.

La classe “**Run**” è classe concreta che modella un allenamento di corsa, eredita da “Endurance” e implementa i metodi virtuali puri:

- **CaloriesBurned()**: definito nella classe “Training”, calcola il numero di calorie bruciate;
- **clone()**: definito anch'esso nella classe “Training”, implementa il “clone pattern” (è un pattern per ottenere un “costruttore di copia polimorfo”).

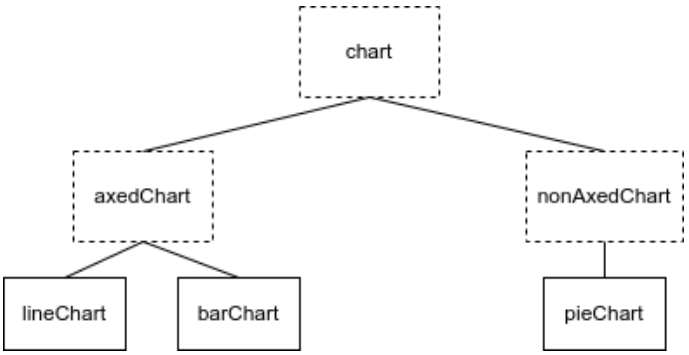
Le classi “**Cycling**” e “**Walk**” modellano tipi di allenamento diversi ma implementano gli stessi metodi virtuali puri, ereditando anch’essi da “Endurance”.

La classe “**Tennis**” modella un allenamento di Tennis, deriva da “Repetition” e implementa i metodi virtuali puri:

- **CaloriesBurned():** definito nella classe “Training”, calcola il numero di calorie bruciate;
- **clone()**: definito anch'esso nella classe “Training”, implementa il “clone pattern” ;
- **Intensity()**: definito in “Repetition”, calcola l’intensità dell’allenamento svolto basandosi su una formula che usa il tempo “attivo” ed il tempo di “riposo” dell’allenamento.

La classe “**Rugby**” è equivalente a “Tennis”, modellando un allenamento della specialità omonima.

La seconda gerarchia presente nel progetto appartiene alla vista:



La classe base astratta “**chart**” modella un widget che gestisce un QChartView ed un QChart, widget che modellano il concetto di “modello” di grafico e di “vista” per tale grafico; in “chart” si ha un metodo di get per la vista del grafico.

La classe astratta “**axedChart**” eredita da “chart” e modella un widget che deve gestire la vista di grafici dotati di assi.

La classe astratta ”**nonAxedChart**” eredita anch’essa da “chart” e modella un widget per la gestione della vista di un grafico non dotato di assi.

La classe concreta “**lineChart**” modella un widget per la gestione della vista di un grafico a linee; tale classe implementa i seguenti metodi virtuali puri:

- **connect()**: connette la “serie” di dati al grafico, è metodo di “chart”;
- **setAxes()**: assegna i nomi delle etichette agli assi del grafico, metodo di “axedChart”;
- **addSeries()**: gestisce l’aggiunta di dati nel grafico, metodo di “axedChart”.

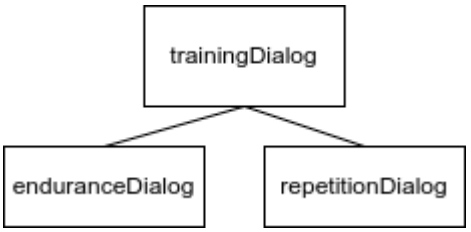
La classe concreta “**barChart**” è equivalente a “lineChart”, modellando però un widget per la gestione della vista di un grafico a barre.

La classe concreta “**pieChart**” modella un widget per la gestione della vista di un grafico a torta; vengono implementati i seguenti metodi virtuali puri:

- **connect()**: connette la “serie” di dati al grafico, metodo di “chart”;
- **addSeries()**: gestisce l’aggiunta di dati nel grafico, metodo di “nonAxedChart”.

NOTA: i metodi riportati non contengono i parametri che effettivamente avrebbero, questo è fatto per evitare di inserire firme di metodi prolisse; i metodi “addSeries()” di “pieChart” e “lineChart” (e “barChart”) hanno parametri diversi.

La terza ed ultima gerarchia appartiene alla vista:



La classe base concreta “**trainingDialog**” implementa un dialog generico per le componenti grafiche comuni tra dialog di gestione di allenamenti di ripetizione e di resistenza.

Il metodo **setupCommon()** inserisce nel layout principale dei widget per la visualizzazione/modifica del nome e dell’inizio di un allenamento.

La classe concreta “**enduranceDialog**” implementa “trainingDialog” aggiungendo dei widget per la visualizzazione della durata dell’allenamento e della distanza percorsa.

La classe concreta “**repetitionDialog**” implementa “trainingDialog” aggiungendo widget (e layout) per visualizzare gli esercizi su cui operare.

Le durate, gli orari, le date e le date-ora sono modellate dalle classi “**TimeSpan**”, “**Date**” e “**DateTime**”, offrendo un’interfaccia semplice per la modifica e le operazioni logiche su tali entità; la classe “**DateTimeConverter**” si occupa della conversione di tali entità logiche in QTime, QDate e QDateTime, ovvero le rispettive classi del framework Qt.

Gli allenamenti sono collezionati all’interno di una **std::list** in quanto essi sono ordinati per data di inizio: le operazioni di **inserimento** e **modifica** devono quindi essere **efficienti** relativamente alle rimozioni ed agli inserimenti “i**n mezzo**” alla collezione, non avendo assicurazioni né sulla posizione in cui tali operazioni avverranno né sul numero di elementi della collezione (a differenza della collezione di esercizi degli allenamenti di ripetizione).

Funzionalità

Le funzionalità offerte dall’applicazione sono le seguenti:

- **Gestione** delle **operazioni** sul piano di allenamenti in modo da far rispettare i vincoli espressi nella sezione “Introduzione”: tali operazioni sono effettuate su una serie di dialog (tra cui quelli specificati nella sezione precedente);
- **Visualizzazione** dei dati immessi per mezzo di **grafici**: l’utente può selezionare dinamicamente il grafico ed i dati da visualizzare su di esso (nel menù “Visualizza” vi è la possibilità di ottenere un dialog ridimensionabile per poter meglio visualizzare i grafici);
- **Gestione** delle entry della “**combo box**” per la selezione dei **dati** da visualizzare sul grafico attualmente visibile: in assenza di determinati tipi di allenamenti (riportati nella sezione precedente) non è possibile visualizzare alcuni tipi di dati e, quindi, il loro riferimento nella “combo box” è nascosto in modo dinamico;
- **Visualizzazione** dei dati immessi per mezzo di **tabelle**: l’utente può scegliere tra due visualizzazioni tabellari distinte, descritte ai punti successivi, in cui gli allenamenti sono ordinati per data di inizio in modo decrescente;
- **Visualizzazione** di una **tabella singola**: se tutti gli allenamenti sono allenamenti di ripetizione (o di resistenza) tale tabella offrirà una colonna in più rispetto al caso in cui tra gli allenamenti vi siano allenamenti sia di resistenza sia di ripetizione (colonna “Distanza” per gli allenamenti di resistenza, “Intensità” per gli allenamenti di ripetizione);
- **Visualizzazione** dei dati immessi per mezzo di due tabelle: la prima tabella conterrà tutti gli allenamenti di ripetizione e la seconda conterrà tutti gli allenamenti di resistenza con le colonne descritte al punto precedente;
- **Visualizzazione** dei dati degli **esercizi** di un determinato allenamento di ripetizione: se vi sono allenamenti di ripetizione nel piano, un pulsante (ed una voce nel menù “Visualizza”) consentono la selezione di un allenamento di ripetizione di cui si vogliono visualizzare i dati degli esercizi;
- **Salvataggio** dei dati in formato XML: ogni piano di allenamento può essere salvato in un file strutturato XML tramite due voci nel menù “File”: “**Salva**” e “**Salva col nome**”; la voce “Salva”, oltre a salvare i dati, consente di poter memorizzare il percorso del file utilizzato in modo da poter, usando lo stesso comando, salvare sullo stesso file una versione aggiornata dei dati senza aprire una seconda volta il dialog di sistema per il salvataggio;

- **Ripristino** dei dati da file XML: è possibile ripristinare, se il file è strutturato in maniera coerente ed i dati sono logicamente corretti, un piano di allenamento precedentemente salvato.

Uso di polimorfismo

Nella gerarchia avente “Training” come classe base sono presenti i seguenti metodi virtuali puri:

- *virtual* TimeSpan *getDuration*() *const* = 0;
- *virtual* unsigned int *CaloriesBurned*() *const* = 0;
- *virtual* Training **clone*() *const* = 0;
- *virtual* ~*Training*() = *default*;
- *virtual* double *Intensity*() *const* = 0 : appartiene a “Repetition”;

La gerarchia avente “chart” come classe base offre i metodi virtuali puri:

- *virtual* void *connect*() = 0;
- *virtual* void *setAxes*(*const* std::string& x, *const* std::string& y)= 0: appartiene ad “axedChart”;
- *virtual* void *addSeries*(*const* std::vector<double>* values, *const* std::vector<DateTime*>* start, bool duration) = 0 : appartiene ad “axedChart”;
- *virtual* void *addSeries*(*const* std::vector<double>* values, bool repetition, bool endurance) = 0 : appartiene a “nonAxedChart”;

La classe “Exercise” offre il seguente metodo virtuale, per consentire l’estensibilità del codice relativamente ad una gerarchia basata su tale classe:

- *virtual* Exercise **clone*() *const*;

Formati di input/output

Come già riportato nella sezione “Funzionalità”, il programma consente di salvare i dati in file strutturati XML; la struttura del file consta dei seguenti tag:

- <**plan**>: indica il piano di allenamenti nel suo complesso;
- <**training**>: indica un allenamento ed ha un attributo di nome “type” che indica il tipo dell’allenamento; sul valore di “type” si decidono i tag da utilizzare per il salvataggio dei dati (con conseguente controllo sulla struttura del file);
- <**name**>: indica il nome di un allenamento;
- <**start**>: indica la data-ora (in formato ISO 8601) di inizio di un allenamento;
- <**duration**>: indica la durata di un allenamento di resistenza;
- <**distance**>: indica la distanza percorsa durante un allenamento di resistenza;
- <**exercises**>: indica una collezione di esercizi di un allenamento di ripetizione
- <**exercise**>: indica un esercizio;

- <**name**> (dentro al tag <**exercise**>): indica il nome di un esercizio;
- <**duration**> (dentro al tag <**exercise**>): indica la durata di un esercizio;
- <**recovery**> (dentro al tag <**exercise**>): indica il tempo di recupero di un esercizio.

NOTA: durante la fase di testing sull'IO si è notato come la traduzione da "double" a "QString" usi il simbolo "." per separare le unità dalla parte decimale su ambiente Windows, a differenza dei sistemi operativi basati su kernel Linux in cui si usa il simbolo ",".

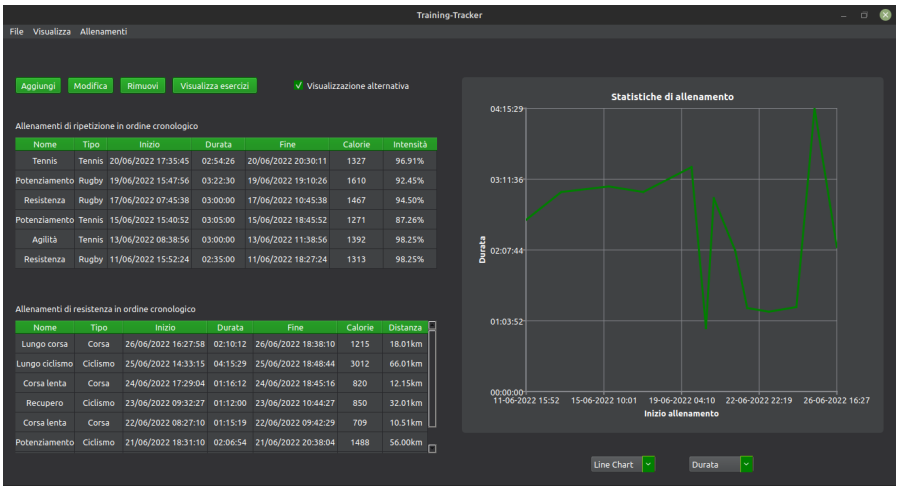
Per questo motivo si è deciso di sostituire il simbolo "," con "." su ambiente Windows, durante la lettura dati (viceversa per tutti gli altri OS).

Manuale d'uso GUI

All'avvio del programma viene visualizzato un **dialog** di **presentazione** dell'applicazione con possibilità di avviare il loop degli eventi (sulla schermata principale della GUI) aprendo o meno un file contenente un piano di allenamenti precedentemente salvato (è fornito il file "dati.xml").

La **schermata principale** è costituita da:

- **Barra dei menù**: sono presenti tre menù:
  - **File**: consente di effettuare operazioni su file e, più in generale, sullo stato dell'applicazione (salvataggio, ripristino, svuotamento del piano di allenamenti, chiusura applicazione);
  - **Visualizza**: consente di visualizzare, per mezzo di dialog, alcuni dati non immediatamente visibili (gli esercizi per esempio) ed i grafici;
  - **Allenamenti**: consente di operare sul piano di allenamenti.
- Widget per la **visualizzazione tabellare**: è costituito da pulsanti dai quali è possibile aprire dialog per operare sugli allenamenti, oltre che da tabelle per la visualizzazione dei dati;
- Widget per la **visualizzazione grafica**: è costituito dal grafico visibile e dalle "combo box" che consentono all'utente di scegliere il tipo di grafico ed i dati da visualizzare.



Istruzioni compilazione ed esecuzione

Per una corretta configurazione, occorre installare i seguenti pacchetti:

- **qt5-default**
- **libqt5charts5-dev**

Nella cartella di consegna è fornito il file "**Training-Tracker.pro**" sul quale richiamare i comandi:
qmake Training-Tracker.pro → *make Training-Tracker.pro* → *./Training-Tracker*

Ore di lavoro richieste

Analisi preliminare:	1 ora
Progettazione modello:	2 ore
Progettazione GUI:	2 ore
Apprendimento libreria Qt:	9 ore
Codifica modello:	15 ore
Codifica GUI:	22 ore
Debugging:	10 ore
Testing:	2 ore
Totale:	**63 ore**

Il totale delle ore di lavoro richieste supera le 50 ore in quanto sono stati riscontrati problemi durante il primo collegamento tra vista e modello e durante la fase di testing: l'uso di "**Valgrind**", strumento di debug per quanto riguarda i problemi di memoria, ha portato alla luce problemi di errata/mancata deallocazione di memoria non facilmente intuibili.

Ambiente di sviluppo

Sistema Operativo:	Linux Mint 20.3 (con desktop environment Cinnamon)
Libreria Qt:	Qt 5.9.5
Compilatore:	g++ 7.5.0