

APPLICAZIONE OROLOGIO

RELAZIONE DESCRITTIVA DEL PROGETTO RELATIVO AL CORSO: PROGRAMMAZIONE AD OGGETTI

Bettin Michele 2042883 | Favaron Riccardo 2042386 | Anno Accademico: 2022/2023

Sommario

[1. INTRODUZIONE 2](#_Toc127062042)

[1.1 PRESENTAZIONE PROGETTO 2](#_Toc127062043)

[1.2 IDEA GENERALE 2](#_Toc127062044)

[1.3 ISTRUZIONI PER COMPILAZIONE ED ESECUZIONE 2](#_Toc127062045)

[1.4 SVILUPPATO E TESTATO SUI SEGUENTI SISTEMI 2](#_Toc127062046)

[1.5 PRIMO AVVIO 3](#_Toc127062047)

[2. MANUALE UTENTE 4](#_Toc127062048)

[2.1 OROLOGIO CON FUSO ORARIO 4](#_Toc127062049)

[2.2 SVEGLIA 5](#_Toc127062050)

[2.3 TIMER 5](#_Toc127062051)

[2.4 CRONOMETRO 6](#_Toc127062052)

[3. DESCRIZIONE DEL MODELLO 6](#_Toc127062053)

[3.1 LA GERARCHIA 6](#_Toc127062054)

[3.1.1 LE BASI 6](#_Toc127062055)

[3.1.2 LE DERIVATE 7](#_Toc127062056)

[3.2 CLASSI DI UTILITÀ 7](#_Toc127062057)

[3.3 USO DELLE CHIAMATE POLIMORFE E DI METODI VIRTUALI 7](#_Toc127062058)

[4. PERSISTENZA DEI DATI 8](#_Toc127062059)

[4.1 FUNZIONALITÀ CHE SI VOLEVA IMPLEMENTARE 8](#_Toc127062060)

[4.2 COME SI È DECISO DI IMPLEMENTARE 8](#_Toc127062061)

[4.3 CHIARIMENTI PER L’USO 9](#_Toc127062062)

[5. MANUALE UTENTE DELLA GUI 9](#_Toc127062063)

[5.1 SHORTCUT DA TASTIERA 9](#_Toc127062064)

[6. METODOLOGIE DI SVILUPPO PROGETTO 10](#_Toc127062065)

[6.1 IDEA DI PROGETTO 10](#_Toc127062066)

[6.2 DEFINIZIONE della gerarchia 10](#_Toc127062067)

[6.4 COLLABORAZIONE 10](#_Toc127062068)

[7.1 RENDICONTO ORE 11](#_Toc127062069)

# INTRODUZIONE

## 1.1 PRESENTAZIONE PROGETTO

L’Applicazione realizzata è un orologio multifunzione per ambienti desktop. Essa è stata realizzata con il linguaggio C++, in particolare con Qt: la libreria multipiattaforma per lo sviluppo di programmi con interfaccia grafica, ormai diventata uno standard e ampiamente presentata durante il corso di Programmazione Ad Oggetti tenuto dal Prof. Ranzato e dal Dott. Zanella.

## 1.2 IDEA GENERALE

L’applicazione che si è deciso di realizzare è un semplice orologio multifunzione che offre all’utente diverse funzionalità riguardanti la gestione del tempo: un orologio con data disponibile con diversi fusi orari, un cronometro, uno o più timer personalizzabili e infine una o più sveglie gestite contemporaneamente. È inoltre disponibile la comoda funzione di salvare e su disco e caricare successivamente un set di sveglie.

La filosofia che si è deciso di adottare per lo sviluppo del software, grazie all’utilizzo dell’ereditarietà e del polimorfismo, è quella dell’estensibilità. Per motivi legati alle tempistiche del progetto, il software proposto risulta snello, ma il codice appunto permette una agevole introduzione di nuovi concetti e funzionalità, tramite futuri aggiornamenti, riutilizzando le parti di codice già sviluppate oppure estendendo quelle esistenti.

## 1.3 ISTRUZIONI PER COMPILAZIONE ED ESECUZIONE

Per compilare Orologio eseguire i comandi:

* qmake
* make

Per eseguire Orologio eseguire il comando:

* ./Orologio

o eventualmente fare doppio clic sul file “Orologio” appena generato dai comandi precendenti.

## 1.4 SVILUPPATO E TESTATO SUI SEGUENTI SISTEMI

|  |  |  |
| --- | --- | --- |
| **Sistema Operativo** | **Versione libreria Qt** | **Versione compilatore** |
| Windows 10 | Qt 6.4.1 | MinGW 11.2 |
| Ubuntu 22.04 (virtual machine) | Qt 6.4.1 | GCC 7.4.0 |
| MacOS Monterey 12.6 | Qt 6.4.1 | Apple clang 14.0.0 |

## 1.5 PRIMO AVVIO

All’apertura del software notiamo che l’interfaccia utente è suddivisa in tre sezioni.

Abbiamo un menù generale nella parte superiore, che ci permette di interagire con la sezione legata alla persistenza dei dati (dunque relativa al salvataggio o all’apertura di dati salvati su disco), di passare da una vista all’altra, accedere alle impostazioni e infine di uscire dall’applicazione.

Successivamente abbiamo un menù laterale che ha il semplice scopo di passare da una funzionalità all’altra del nostro Orologio (compresa la schermata Impostazioni) con dei comodi ed intuitivi bottoni forniti di icona.

Successivamente abbiamo la schermata di destra (che occupa la maggior parte dello schermo) dove si può interagire con le vere e proprie funzionalità: orologio, cronometro…etc.



# MANUALE UTENTE

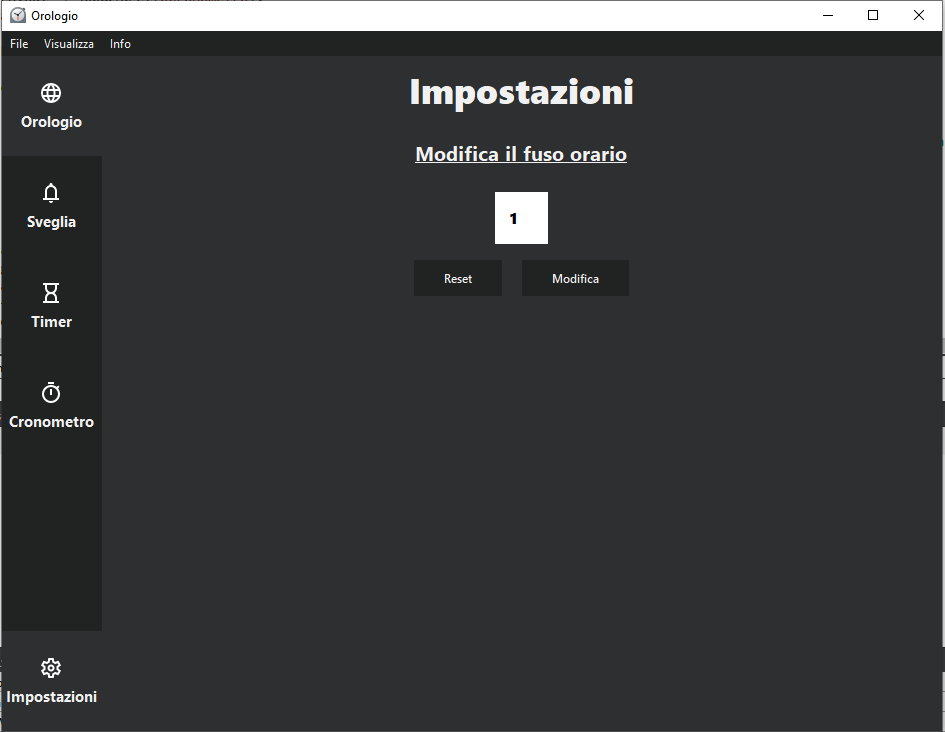
Di seguito descriviamo brevemente le funzionalità proposte dall’applicazione:

## 2.1 OROLOGIO CON FUSO ORARIO

In questa sezione è reso disponibile all’utente un orologio singolo con data relativo ad uno specifico fuso orario, come indica la foto sopra. Quest’ultimo (il fuso orario) al primo accesso viene dedotto dalla sottrazione tra l’orario UTC e l’orario del computer dove questo programma viene eseguito.

Successivamente, esso può essere manualmente settato dall’utente nell’applicazione recandosi sulla sezione Impostazioni > Fuso Orario > Modifica il fuso Orario.

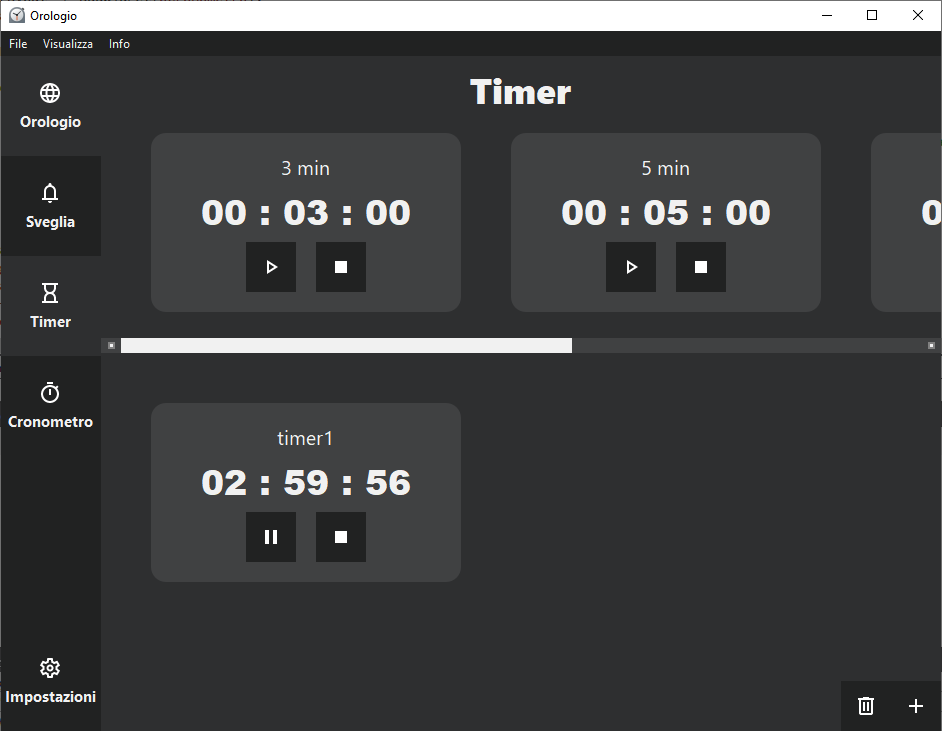
Di seguito un’immagine esplicativa:



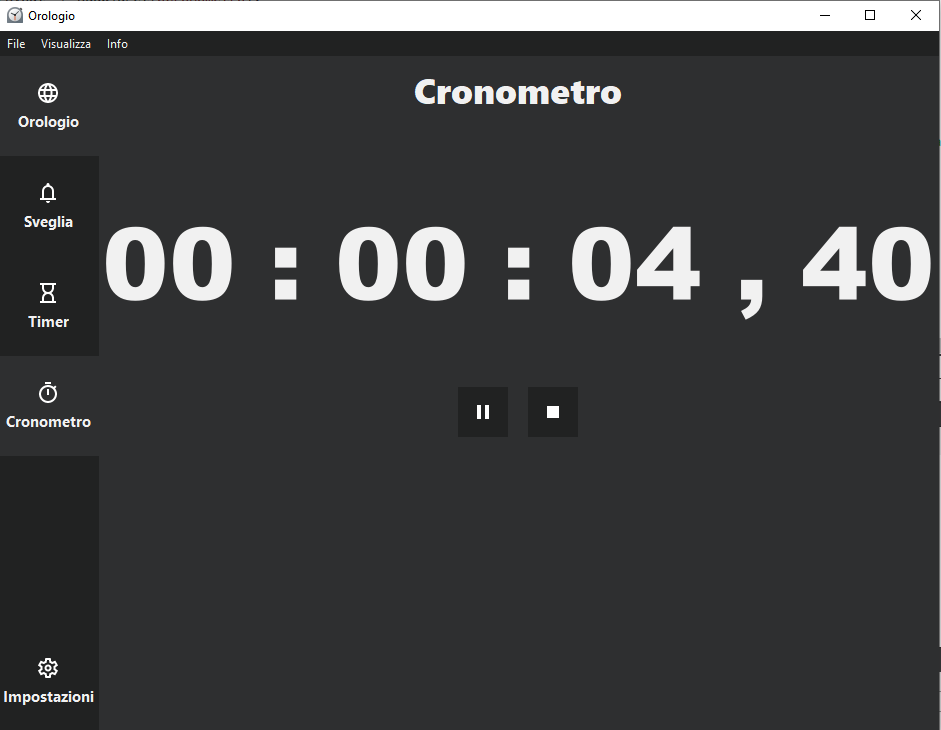
## 2.2 SVEGLIA

In questa sezione è possibile impostare un numero arbitrario di sveglie. Tutto quello che si deve fare per aggiungere una sveglia è inserire l’ora in cui si vuole che la stessa “suoni”. È importante specificare che le sveglie non producono un vero e proprio suono, ma al loro scadere appare un pop up che ne indica lo scoccare. Tutte le sveglie prendono come riferimento l’orario della sezione OROLOGIO per capire quando è la loro scadenza.

## 2.3 TIMER

In questa sezione è possibile impostare un numero arbitrario di timer. Tutto quello che si deve fare per aggiungere un timer è inserire la durata dello stesso in termini di: ore, minuti e secondi. È importante specificare che lo scadere dei timer non produce un vero e proprio suono, ma un pop up che ne indica l’esaurimento. Tutti i timer prendono come riferimento l’orario della sezione OROLOGIO per capire quando è la loro scadenz

## 2.4 CRONOMETRO

In questa sezione è possibile avviare un cronometro, con precisione al centesimo di secondo. Tutto quello che si deve fare è utilizzare i pulsanti di inizio, pausa e azzera per utilizzare lo strumento offerto.

# DESCRIZIONE DEL MODELLO

Il nostro modello dati è costituito da una gerarchia composta da 6 classi (la quale sfrutta, relazione has-a, ereditarietà, ereditarietà multipla e il polimorfismo). Successivamente sono presenti altre classi di utilità utili all’implementazione delle funzionalità del progetto.

## 3.1 LA GERARCHIA

La base della nostra gerarchia è costituita da due classi concrete Orario e Data. Successivamente altre quattro classi derivate ereditano da una di queste due o entrambe.

## 3.1.1 LE BASI

Orario costituisce un semplice orario nel formato: ore, minuti e secondi con i vari metodi di utilità per modificare, visionare e aggiornare le informazioni relative allo stesso.

Data rappresenta una semplice data nel formato: giorno, mese e anno (dove mese è visualizzato successivamente con il nome in italiano dello stesso) con i vari metodi di utilità per modificare, visionare e aggiornare le informazioni relative all’oggetto presente.

### 3.1.2 LE DERIVATE

Orologio costituisce l’unione di Orario e Data in un'unica classe aggiungendo l’informazione relativa al fuso orario che determinerà il cambiamento dell’orario ed eventualmente della data di una quantità di ore proporzionale al fuso specificato.

Cronometro estende Orario, il quale lo inizializza all’ora 00:00:00 e successivamente ne incrementa il valore ogni secondo o mette a disposizione i relativi metodi di pausa o azzeramento.

Timer estende Orario, il quale lo inizializza con la durata dello stesso. Questo oggetto particolare avrà a disposizione un particolare metodo di aggiornamento che al posto di incrementare l’orario andrà a decrementarlo ogni secondo, decretandone il termine al raggiungimento dell’ora 00:00:00

Sveglia estende Orario e implementa una relazione has-a con la classe Orario. Si inizializza uno dei due orari con l’orologio globale e l’altro con la scadenza della sveglia che si intende impostare. Successivamente l’orologio globale (che è aggiornato ogni secondo) comparerà sé stesso con lo scadere della sveglia e qualora questi due combaciassero si fa apparire il pop up di fine sveglia e si termina la comparazione.

## 3.2 CLASSI DI UTILITÀ

All’interno della gerarchia sono presenti due ulteriori classi:

CallBackTimer che permette la schedulazione di un’attività descritta dall’utente (passata come parametro sottoforma di funzione lambda) ogni quanto di tempo specificato. Questa classe è necessaria per implementare tutte le funzioni che richiedono un aggiornamento continuo o una comparazione di orari continua ogni secondo o centesimo di secondo.

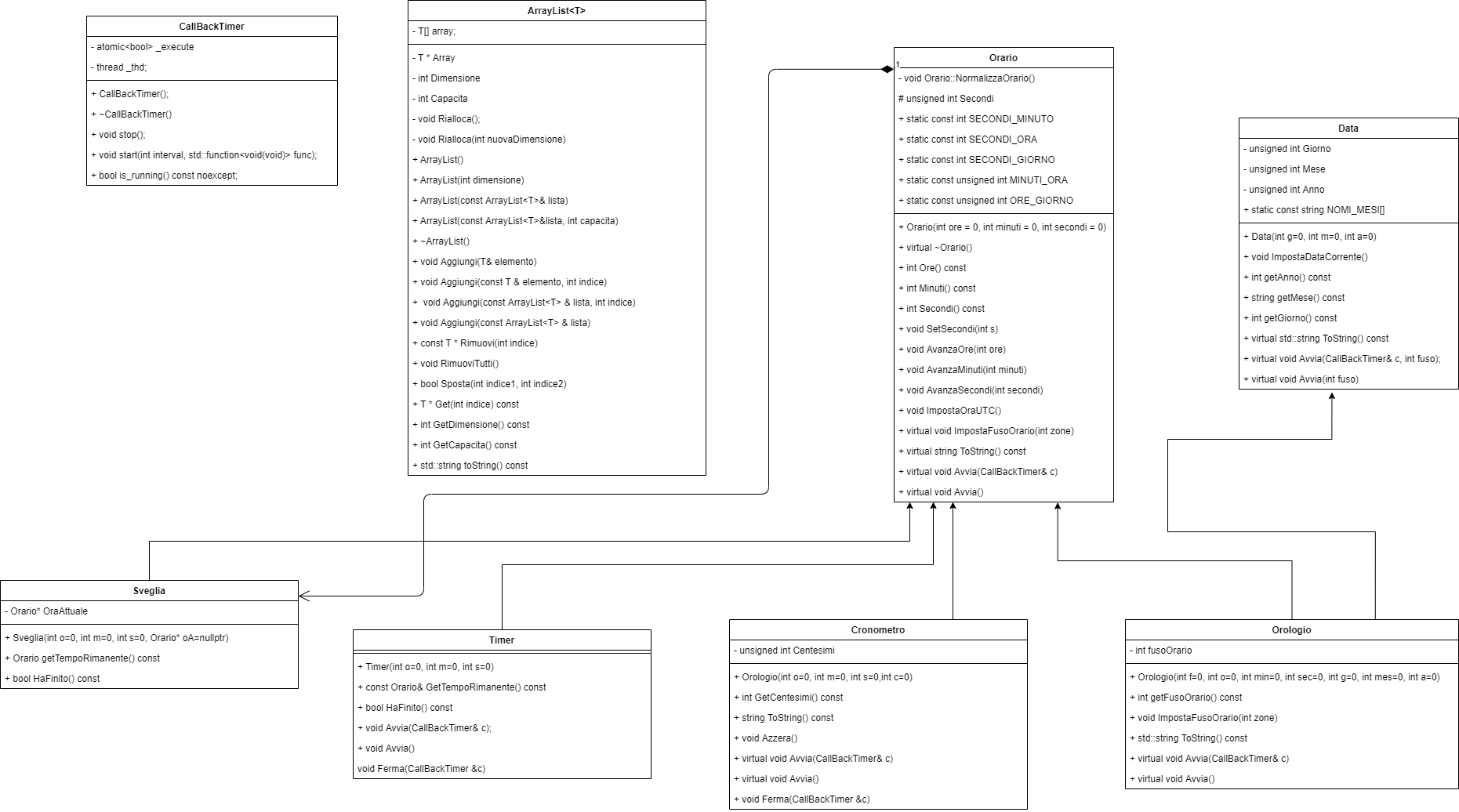
ArrayList è un vettore dinamico, quindi una struttura dati array che può essere ridimensionata e consente di aggiungere o rimuovere elementi. È fornita con la libreria standard di C++ ma per scopi accademici si è deciso di introdurre una nostra implementazione.

## 3.3 USO DELLE CHIAMATE POLIMORFE E DI METODI VIRTUALI

Ogni classe appartenente alla gerarchia possiede un metodo virtuale chiamato Avvia(), disponibile anche con diversi overloading. A tal fine, ogni oggetto della gerarchia che si sta trattando, indipendentemente dal suo tipo dinamico può essere aggiornato (una tantum con Avvia() o costantemente ogni quanto di tempo con Avvia(CallBackTimer)).

Questo approccio può risultare particolarmente utile qualora dovessimo operare su collezioni di oggetti con tipo statico fisso ma tipo dinamico indefinito, così facendo per ogni oggetto possiamo invocare i metodi richiesti ed essere sicuri che verrà invocato a run-time la corretta implementazione del metodo.

Di seguito si presenta una rappresentazione del diagramma UML delle classi relative al modello dati:



# PERSISTENZA DEI DATI

## 4.1 FUNZIONALITÀ CHE SI VOLEVA IMPLEMENTARE

L’idea di base era quella di permettere all’utente di potersi salvare una configurazione particolare di sveglie (dopo averle create manualmente ovviamente) e poterne fruire successivamente in futuro grazie ad una particolare funzione che permettesse di aprire tale set di sveglie.

Per motivi legati al rendere il tutto più semplice possibile abbiamo fatto si che l’utente non possa salvare più configurazioni su file diversi o sullo stesso file.

Inoltre abbiamo esentato l’utente dalla scelta grafica, grazie al file system di window del file dove salvare le informazioni relative al set di sveglie. Tutto verrà fatto automaticamente grazie ai pulsanti salva e apri posti sul menù nella parte superiore dell’applicazione.

## 4.2 COME SI È DECISO DI IMPLEMENTARE

Per rimanere nell’intervallo di ore richiesto, siamo andati a creare un semplicissimo parser che grazie ad alcune specifiche classi della libreria Qt (capaci appunto di manipolare file e stream di dati) mettesse a disposizione due funzionalità.

La prima si chiama scriviArraySuFileJson. Essa è capace di scrivere su un file json e quindi rendere persistente le informazioni relative ad un set di sveglie precedentemente create manualmente, una volta specificato il tipo degli oggetti che stiamo trattando (Widget relativi alle Sveglie nel nostro specifico caso), un percorso file relativo ad un file con estensione .json e un ArrayList di oggetti del tipo specificato.

La seconda si chiama caricaArrayDaFileJson. Essa è capace di popolare un ArrayList rappresentante un set di sveglie a partire da un file .json che è stato precedentemente popolato, una volta specificato il tipo degli oggetti che stiamo trattando (Widget relativi alle Sveglie nel nostro specifico caso) e un percorso file relativo ad un file con estensione .json.

## 4.3 CHIARIMENTI PER L’USO

Se si ha già creato delle sveglie manualmente dall’interfaccia e si importa una configurazione con il bottone “apri” si perderanno automaticamente tutte le sveglie create fino a quel momento, sovrascrivendo la lista con le nuove importate da file.

Una volta importata una configurazione è possibile concatenare alla seguente nuove sveglie creandole manualmente dall’interfaccia. Successivamente se si vuole si può tranquillamente salvare su disco questa nuova configurazione aggiornata.

# MANUALE UTENTE DELLA GUI

Le funzionalità del programma sono molto user friendly e facilmente individuabili grazie ad un appropriato uso delle icone. Si seguito si discute semplicemente delle shortcut da tastiera presenti che rendono alcune funzioni comuni ancora più rapide.

## 5.1 SHORTCUT DA TASTIERA

L’applicazione presenta un vasto uso delle shortcut da tastiera per favorire una maggiore user friendly experience.

Sono presenti le seguenti shortcut da tastiera:

|  |  |
| --- | --- |
| **Shortcut** | **Spiegazione** |
| Ctrl + O | Apertura file contenente le sveglie salvate |
| Ctrl + S | Salva le sveglie create all’interno del file adibito a tale scopo |
| Ctrl + Q | Comando per uscire dall’applicazione Orologio |
| Ctrl + 1 | Cambia la vista su Orologio |
| Ctrl + 2 | Cambia la vista su Sveglia |
| Ctrl + 3 | Cambia la vista su Timer |
| Ctrl + 4 | Cambia la vista su Cronometro |
| Ctrl + 0 | Cambia la vista su Impostazioni |
| Ctrl + I | Apre la finestra dialog contenente le informazioni sull’app |

# 6. METODOLOGIE DI SVILUPPO PROGETTO

## 6.1 IDEA DI PROGETTO

Per realizzare questo progetto ci siamo lasciati ispirare dall’interfaccia grafica e dalle funzionalità offerte da alcune web app e dall’applicazione dell’orologio di Microsoft Windows.

## 6.2 DEFINIZIONE della gerarchia

Per sviluppare la gerarchia costituita dai vincoli di progetto abbiamo realizzato prima un diagramma UML delle classi (anche se non abbiamo ancora frequentato il corso Tecniche e Progettazione per lo Sviluppo Software) che ci ha permesso di individuare agevolmente le varie dipendenze dettate da una buona programmazione con paradigma orientato agli oggetti. Il diagramma UML è stato disegnato con il software gratuito Draw.io.

6.3 CREAZIONE DELLO STORYBOARD

Per sviluppare la GUI abbiamo utilizzato Figma, questo software ci ha permesso di abbozzare agevolmente tutta la grafica necessaria per implementare tutte le funzionalità che permetteva di sfruttare il modello dati realizzato.

## 6.4 COLLABORAZIONE

Per sviluppare il codice dell’applicazione ed avere sempre una corretta sincronizzazione il mio lavoro e quello del mio compagno ci siamo affidati al famoso software di versionamento Git. Ed abbiamo usato GitHub per gestire la repository.

Abbiamo inoltre usato un software in stile GannProject per gestire i singoli task di ognuno di noi, le milestone e le relative tempistiche.

Io e il mio compagno abbiamo collaborato interamente per le cinquanta ore relative al progetto grazie ad un software di videochat chiamato Discord che ci permetteva di parlarci e proiettare il nostro schermo a vicenda.

# 7.1 RENDICONTO ORE

|  |  |  |
| --- | --- | --- |
| **Attività** | **Ore Previste** | **Ore Effettive** |
| Studio e progettazione | 10 | 15 |
| Sviluppo del codice del modello | 10 | 9 |
| Studio del framework Qt | 10 | 10 |
| Sviluppo del codice della GUI | 10 | 9 |
| Test e debug | 5 | 8 |
| Stesura della relazione | 5 | 5 |
| **totale** | 50 | **56** |

Il monte ore è stato superato di sei unità, questo è stato causato da eccessiva attenzione nella fase di studio e progettazione. Inoltre, i tempi in questa fase sono stati leggermente allungati in quanto inizialmente si era pensato allo sviluppo di un’altra idea: la progettazione di un software per la gestione base delle giacenze di un magazzino. Successivamente ci si è ben presto resi conto dell’impossibilità di realizzare in modo soddisfacente tale realtà in 50 ore data la complessità del modello sottostante.