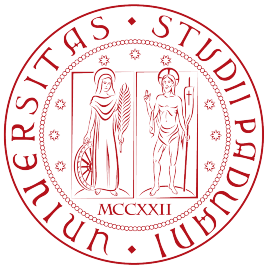


UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Matematica Tullio Levi-Civita
Corso di Laurea Triennale in Informatica

Relazione del progetto "QtSalaries"

Bulbarelli Giacomo
1144046

Anno Accademico 2019/2020

Abstract

Relazione sullo sviluppo del progetto "QtSalaries" per il progetto del corso di Programmazione ad Oggetti svoltosi durante l'anno accademico 2019/2020 presso la facoltà di Informatica dell'Universita' degli studi di Padova.
Il progetto prevede la creazione di una gerarchia di tipi (che utilizza la derivazione multipla e il polimorfismo) e di un container templatizzato che permette l'utilizzo della gerarchia creata, unita alla realizzazione di un interfaccia grafica realizzata utilizzando il framework *Qt*.
Il tutto viene proposto come complemento alla prova scritta del corso e serve a verificare l'effettiva comprensione del paradigma di programmazione ad oggetti (applicato al linguaggio di programmazione *C++*) contestualizzata in uno scenario applicativo quanto più possibile vicino a quello di uno sviluppo software in ambienti reali. Il lavoro è stato svolto dal relatore di questa relazione e dalla studentessa

Veronica Barbieri(Matr. 1143463)

che ha partecipato allo sviluppo in ogni sua fase, compatibilmente alle restrizioni nell'interazione imposte dall'emergenza sanitaria da pandemia del COVID-19

Contents

1 Introduzione: Idea di progetto

L'applcativo *QtSalaries* nasce con l'intento di fornire uno strumento per il supporto al calcolo degli stipendi mensili nell'ambito di aziende che si occupano della gestione di catene di supermercati, lasciando contemporaneamente aperta la possibilità di essere esteso a qualunque tipo di attività aziendale. Detta più semplicemente, un libro paga elettronico.

Nella realtà da noi immaginata, l'azienda possiede un numero non precisato di punti vendita che sono composti dalla presenza di:

- 1 Direttore (con contratto Full-time e bonus)
- n impiegati con contratto Full-time
- n impiegati con contratto Part-time

con n numero intero non rigidamente limitato.

Inoltre, per ciascun mese di lavoro vengono fornite le ore di lavoro svolte dal dipendente (ore regolari e straordinarie) e i giorni lavorativi effettivi. **N.B.**: nella realtà da noi immaginata, un giorno lavorativo straordinario non viene contata come giornata lavorativa effettiva, ma viene tenuto traccia delle ore di lavoro extra che vengono poi sommate al totale delle ore lavorate. Quindi un intera giornata di lavoro viene vista come 8 ore di straordinario, e non viene conteggiata nei giorni di lavoro effettivi. Con questa idea in mente, si è stabilito in maniera congiunta di procedere inizialmente con lo sviluppo della gerarchia di tipi e del container, ignorando momentaneamente lo sviluppo di un interfaccia utente.
Tale scelta nasce dalla volontà di creare una divisione netta tra il modello logico dell'applicazione e l'interfaccia utente, rispettando quindi il vincolo di separazione tra i due imposto dalle specifiche di progetto fornite dal professore.

2 Gerarchia di tipi

2.1 Classi base

La gerarchia di tipi di compone di due classi base astratte: `worker` e `contract` che fungono da base per l'intera gerarchia di tipi.
In particolare:

- **worker** definisce le caratteristiche elementari di un lavoratore qualunque.
- **contract** definisce una base per i vincoli contrattuali che devono essere definite da tutte le classi che derivano da essa e a cui un lavoratore deve riferirsi.

Cercando di dare una visione più complessiva di questa intuizione, un lavoratore può essere visto come un entità a cui vengono associate una persona (worker) e i vincoli contrattuali a cui quella persona deve tener fede (contract). Verranno di seguito esaminate le caratteristiche di entrambe le classi in maniera più dettagliata.

2.1.1 Classe worker

La classe worker fornisce le funzionalità di base per lo store delle informazioni relative ad un dipendente aziendale qualsiasi e fornisce la firma per i metodi necessari al calcolo degli stipendi, dichiarandoli come virtuali puri e delegandone l'onere implementativo alle classi figlie.
La scelta viene giustificata dal fatto che uno stipendio per un dipendente non può essere calcolato se non tenendo in considerazione quelli che sono i vincoli contrattuali a cui il dipendente stesso è sottoposto, definiti in maniera precisa dalla classe contract.

2.1.2 Classe contract

La classe contract fornisce i vincoli per i dati essenziali relativi alle tipologie di contratto proposte dall'azienda. Essa deve fungere da base per la definizione dei contratti specifici, e per questo prevede unicamente la definizione di metodi virtuali puri che vincolano le classi figlie a fornire un set di informazioni elementari sulla categoria di contratto che rappresentano. Tali informazioni sono:

- Le ore lavorative previste per la singola giornata di lavoro;
- Il salario orario base per il contratto (un retribuzione fissa oraria);
- Il salario bonus per il contratto (una retribuzione extra da sommare al salario base orario per ciascuna delle ore di straordinario eventuali lavorate);
- I giorni di ferie mensilmente guadagnati da un dipendente;

2.2 Classi derivate di Contract

Le classi derivate da contract rappresentano in tutto e per tutto le tipologie di contratto fornite dall'azienda.
Nel nostro esempio specifico, vengono proposte due tipologie di contratto:

- **Part-time** (level0) si presenta come un contratto part-time classico, pensato per i nuovi assunti nella realtà aziendale e destinato ad essere evoluto (dopo che il neo-assunto ha aquisito sufficiente dimestichezza con il proprio ruolo) in un contratto Full-Time
- **Full-time** (level1) si presenta come il contratto standard per i dipendenti dell'azienda.

Vengono di seguito riportati i dati specifici contenuti in ciascuna delle due classi in forma tabulare

Part-time

Parametro	**Valore**
Ore lavoro giornaliero	4 ore
Compenso orario	9 euro
Bonus ore straordinario	0.5 euro
Giorni feriali accumulabili mensilmente	1 giorno

Full-time

Parametro	Valore
Ore lavoro giornaliero	8 ore
Compenso orario	8 euro
Bonus ore straordinario	1.0 euro
Giorni feriali accumulabili mensilmente	1 giorno

N.B.: l'azienda vincola i propri dipendenti a poter avere un unico turno della durata prevista dal contratto per giorno. Quindi un dipendente Part-time lavorerà 4 ore al giorno massime, mentre un dipendente Full-time lavorerà 5 giorni la settimana per 8 ore al giorno massime. Tale cifra non fa riferimento alle ore di lavoro straordinare che possono essere richieste al lavoratore, ma solamente a quelle "standard"

2.3 Classi derivate da worker e contract

Le classi descritte in seguito rappresentano il punto di arrivo di questa gerarchia. Sono infatti le uniche classi che a conti fatte verranno utilizzate per lo store delle informazioni, e rappresentano un dipendente nella sua interezza. Esse sono complessivamente 3, ciascuna volta a rappresentare una delle tipologie di dipendente che un punto vendita può presentare (di quelli descritti nella Sezione 1).

- **Part-time employee** (ptemployee) rappresenta un dipendente part-time
- **Full-time employee** (ftemployee) rappresenta un dipendente full-time
- **Director** (director) rappresenta un dipendente che svolge il ruolo di direttore

Ciascuna classe contiene dei parametri che si vanno ad aggiungere a quelli del contratto specifici del ruolo che il dipendente ricopre all'interno dell'azienda. Per quanto questa scelta progettuale sia facilmente classificabile come imprecisa (sarebbe stato altrettanto valido creare delle classi contratto specifiche per ogni ruolo offerto dall'azienda) diventa necessaria per poter applicare a ruoli differenti la stessa base contrattuale. Il tentativo è quello di rendere le classi figlie di contract quanto più generali e riutilizzabili possibile, andando a definire le specificità del ruolo dentro la classe che definisce il ruolo stesso. I suddetti dati sono riportati in forma tabulare qui sotto.

Part-time employee

Nessun bonus

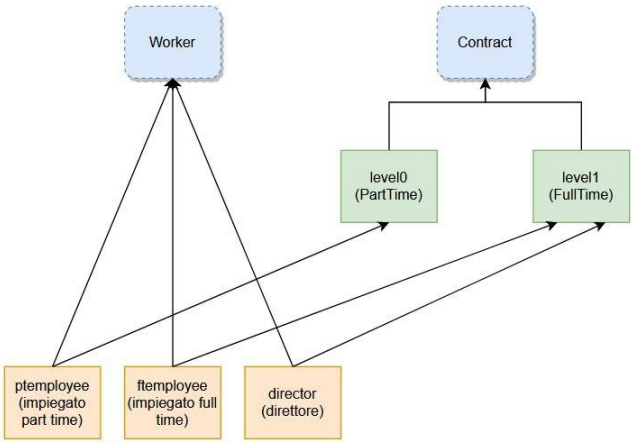
Full-time employee

Parametro	Valore
Salario base bonus	100 euro
Giorni ferie bonus	1 giorno

Full-time employee

Parametro	Valore
Salario base bonus	400 euro
Giorni ferie bonus	1 giorno
Bonus aggiuntivo su ore straordinario	1.5 euro

Di seguito viene riportata un immagine che propone una visione d'insieme della gerarchia proposta, vista nella sua interezza.



3 Container

Il container consiste in un template che implementa una lista doppiamente linkata. Utlizza un sotto-tipo nodo per immagazzinare le informazioni e mette a disposizione dell'utente un const iterator per lo scorrimento della lista. Lo sviluppo di questa sezione è stato portato avanti dalla collega, che l'ha realizzato prendendo ispirazione dagli esempi portati durante il corso dal professore e dal classico `std::vector` presente nella libreria STL.

4 Model / Classe Payroll

Il model si presenta come una classe implementa il container realizzato istanziato a puntatori a worker (classe base della gerarchia) e tutti i metodi necessari alle logiche del programma. E' a tutti gli effetti il modello logico che verrà utilizzato dall'applicazione.E' in questa classe che prende un senso concreto utilizzare l'utilizzo del polimorfismo. Tutti i metodi operano su puntatori a worker, senza preoccuparsi di quale sia il tipo effettivo dell'oggetto puntato al momento di una chiamata di funzione su un oggetto.Le funzionalità presenti sono quelle che verranno poi messe a disposizione anche dalla GUI, coadiuvate da alcune di utilità aggiunte in un secondo momento per facilitare la comunicazione tra le

componenti, senza però vincolare l'utilizzo del modello all'interfaccia proposta e esaminata nel seguito.

4.1 Utilizzo del polimorfismo in dettaglio

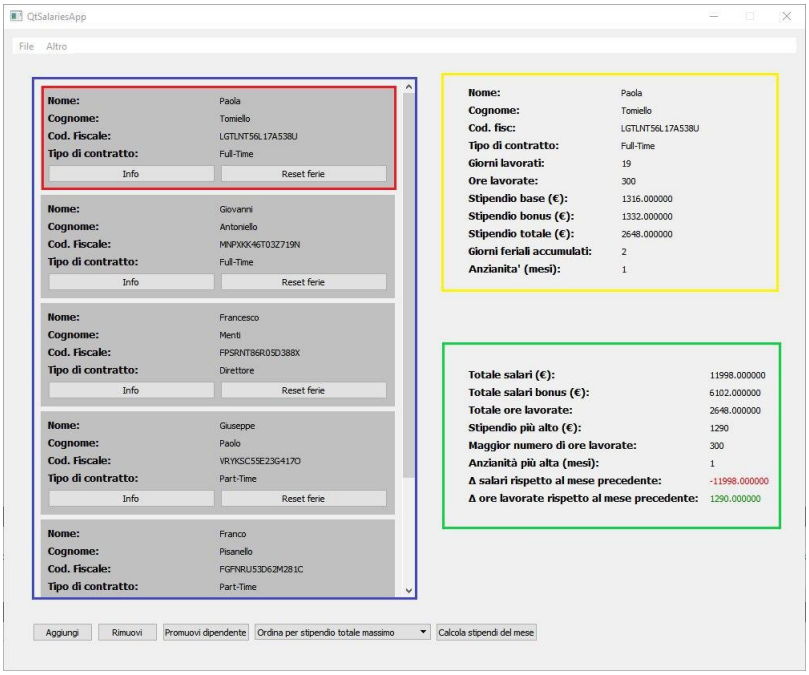
Il polimorfismo risulta necessario al fine di calcolare i parametri salariali di ciascuna tipologia di lavoratore. Tale calcolo infatti dipende fortemente dal tipo di contratto (e quindi dal sotto-oggetto che lo rappresenta) a cui il lavoratore fa riferimento, e da tutti i bonus e le specificità che il ruolo stesso porta con sè. I metodi che si occupano di calcolare questi parametri sono:

```
virtual double calcBaseSal() const =0;
virtual double calcBonus() const =0;
virtual void updateVacAcc() =0;
virtual double calcFullSal(const int&, const int&) =0;
```

Ciascuno di essi utilizza i dati propri della classe e i dati del contratto per il calcolo degli sipendi. Vi è inoltre un metodo virtuale puro nella classe base `contract` utile per recuperare il nome della tipologia di contratto (utile a fini pratici ma sostanzialmente trascurabile nell'ottica dell'utilizzo del polimorfismo). Questo permette di definire sempre nuove tipologie di contratto e di implementare in una classe worker che ne fa uso le specifiche metodlogie di calcolo dei parametri.

5 GUI

La vista è stata sviluppata seguendo il pattern architetturale MVP(Model-view-presenter). Il risultato finale si presenta come segue, con l'highlight di alcuni elementi grafici successivamente discussi



Si è quindi cominciato costruendo i widget che sarebbero poi andati a comporre l'interfaccia utente, per poi passare al collegamento di tutte le funzionalità tramite l'implementazione del controller.

Le funzionalità proposte ricalcano per la gran parte quelle rese disponibili dal modello, quindi Aggiunta (tasto "Aggiungi") e Delezione (tasto "Rimuovi") di un dipendente, promozione del contratto di un dipendente Part-time a contratto Full-time (tasto "Promuovi"), ordinamento della lista secondo tutta una serie di parametri (ComboBox situato tra i pulsanti "Promuovi" e "Calcola stipendi del mese") e calcolo degli stipendi del mese appena trascorso (pulsante "Calcola stipendi del mese"). In aggiunta a queste, vengono fornite anche funzionalità di salvataggio/caricamento da file (formato .json) e creazione di un nuovo libro paga con parametri azzerati. L'interfaccia si compone di:

- La barra del Menù, dalla quale si può accedere alle funzionalità di creazione di un nuovo libro paga, di salvataggio dei dati correnti e del caricamento di dati salvati su un file (file di caricamento e salvataggio sono disponibili unicamente nel formato json);
- Un widget che elenca tutti i dipendenti presenti nel libro paga (classe `qemployeelist` , colore blu in figura
- Un widget che, all'interno della lista, si occupa di fornire le infromazioni base su un dipendente e due pulsanti di utilità (classe `qemployee`, colore rosso in figura)
- Un widget che mostra i dati completi relativi ad un impiegato (`qemployeeinfo`, colore giallo in figura)
- Un widget che mostra tutti i dati relativi al libro paga nel suo complesso (`qpayrollinfo`, colore verde in figura)
- Un serie di pulsanti nella parte inferiore dello schermo per interagire/modificare i dati contenuti nel libro paga.

In particolare, per ciascuno dei pulsanti (ad eccezione del `QComboBox`) è stato costruito un widget da mostrare in un dialog quando uno di essi viene premuto. Essi consistono in:

- Per il pulsante "Aggiungi" il widget della classe `qadddialog`
- Per il pulsante "Rimuovi" il widget della classe `qdeldialog`
- Per il pulsante "Promuovi dipendente" il widget della classe `qpromodialog`
- Per il pulsante "Calcola stipendi del mese" il widget della classe `qemployeelistforcalc` che similmente a quanto accade in `qemployeelist` ha un sotto-widget che rappresenta il singolo impiegato di cui devono essere inseriti i dati (giorni lavorati nell'ultimo mese e ore lavorate nell'ultimo mese) rappresentato dalla classe `qemployeeforcalc`

La motivazione di un cosi' vasto impegno nella realizzazione di tutti questi widget è la volonta di rendere quanto più modulare possibile ciascuna componente dell'interfaccia grafica, accentuandone la riusabillità.

5.1 Controller

Il controller è stato realizzato seguendo tutte le linee guida esposte dal tutor nelle videolezioni, in maniera equa da entrambi i componenti del gruppo che lo aggiornavano mano a mano che si rendevano necessarie funzionalità specifiche al fine di collegare tutte le componenti.
In tutta semplicità, esso consiste di un riferimento alla vista e un riferimento al modello, e quando la vista notifica la volontà dell'utente di modifcare lo stato del modello, il controller aggiorna modello e vista conseguentemente.

6 Formato I/O su file

Per l'I/O su file si è optato per l'utilizzo del formato `json` principalmente perchè la libreria Qt fornisce tutti gli strumenti per la manipolazione di file in questo formato e data la diffusione importante di questa tipologia di file. Abbiamo valutato la necessità di una funzionalità di salvataggio/caricamento perchè calzante nell' idea dell'applicativo (un libro paga incapace di tener traccia dei dati relativi ai dipendenti è, a conti fatti, inutile). La codifica è stata svolta dalla collega nelle fasi finali di sviluppo, con il mio aiuto sulla risoluzione di cavilli implementativi.

7 Exception handling

Questa parte del progetto è stata curata dalla collega, almeno per quello che concerne la pura codifica. Dopo una breve discussione sul tema, si è deciso di utilizzare le eccezioni per bloccare l'input dell'utente laddove questi potevano risultare "dannosi" e per controllare gli input forniti/avvisare l'utente dell'avvenuta inserzione di dati non coerenti o illegittimi (es. Delezione di un dipendente non presente nel libro paga).

8 Dati sullo sviluppo

8.1 Tempo di lavoro

I tempi complessivi di sviluppo sono riportati nella tabella seguente, con i dati espressi in ore

Task	**Tempo speso**
Progettazione	10h
Container + gerarchia	10h
Apprendimento iniziale Qt	10h *
UI	20h
Controller View e Collegamento senza salvataggio ed eccezioni	30h *
Eccezioni	15h *
I/O	10h
Testing e Debug	10h *

I tempi presentati sono approssimativi dei tempi totali. Le tempistiche marcate con * indicano che in quella fase di sviluppo sono stati incontrati molti più

problemi di quelli previsti, e i tempi di sviluppo hanno subito una dilatazione considerevole.

8.2 Versioni software e istruzioni di installazione

Il progetto è stato sviluppato per conto mio su una macchina che monta una distribuzione linux Pop! OS 19.10, compilatori clang ver. 9.0.0-2 e GCC 64-bit ver. 9.2.1 20191008 (in momenti diversi) e versione Qt 5.9.5 Menzione speciale al sistema di versionamento Git e alla piattaforma GitHub. Senza di essi, saremmo stati perduti.
Per la corretta compilazione del codice sulla macchina virtuale fornita, devono essere eseguiti i seguenti comandi

```
1 qmake -project "QT += widgets"
2 qmake
3 make
4 ./nome_eseguibile_prodotto
```

Viene inoltre fornito un file di esempio in formato json che contiene un sample di dati da utilizzare per il test (nome del file: "payroll.json").

8.3 Suddivisione del lavoro progettuale

La progettazione è stata portata avanti in ogni frangente dai componenti in maniera congiunta, cosi come la realizzazione della gerarchia, del modello e del container. La realizzazione della vista e del controller hanno visto la partecipazione di entrambi in maniera più separata (la codifica di buona parte della vista e il wiring della prima parte di funzionalità erano a carico mio, mentre l'exception-handling, l'I/O su file e il collegamento della vista a queste funzionalità sono stati presi a carico dalla collega). Anche in questo caso le stime sono approssimate e il lavoro è stato accompagnato da un interazione costante tra i componenti. Il mio nome, cognome e numero di matricola si trovano in copertina, quelli della collega nell'abstract.

9 Conclusioni

Il progetto è stato portato avanti ad un ritmo sostenuto, e trova (a mio modo di vedere) la sua più grande vulnerabilità nella gerarchia proposta. Essa è il risultato di un lavoro molto più sofferto di quello che si preventivava (particolarmente spinoso è stato il vincolo sull'ereditarietà multipla, non tanto in fase di codifica o comprensione quanto piuttosto sulle difficoltà nel trovare una realtà da modellare in cui non risultasse eccessivamente forzata una derivazione multipla). Un altro aspetto che ha richiesto particolare sforzo è stato padroneggiare Qt e utilizzare il materiale fornito dagli sviluppatori del framework per l'apprendimento delle basi della libreria. Quello che è innegabile è l'impatto positivo che il progetto ha avuto nella mia personale comprensione della materia e di cosa significhi sviluppare e apprendere una nuova tecnologia. Un ringraziamento speciale và al COVID-19 per non aver permesso al sottoscritto e collega di vedersi come persone civili per tutta la durata del progetto, rendendo un inferno la comunicazione e il coordinamento delle attività. Un grazie di cuore.