Pivato Davide

Matr. 1187595

**FastSalaries**

**Progetto di Programmazione ad Oggetti**

**Anno Accademico 2019/2020**

**Di Davide Pivato e Andrea De Tomasi**

**Relazione di Davide Pivato**

**Indice**

1. Descrizione della Gerarchia

2. Uso del polimorfismo

3. Formato di Input/Output

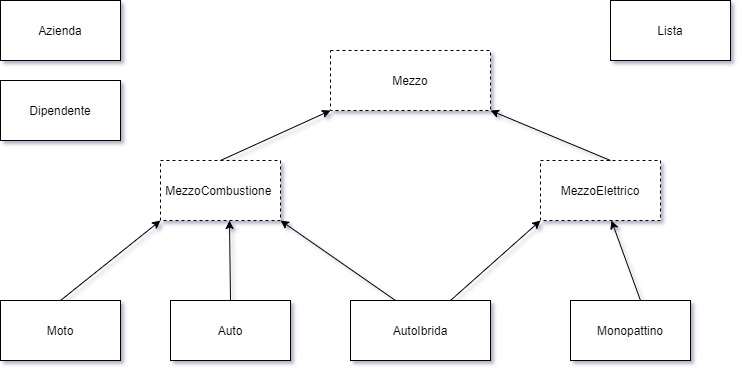
4. Suddivisione del lavoro progettuale e analisi delle ore richieste

5. Manuale Utente della Gui

**1. Descrizione delle gerarchie**

Viene riportato qui sotto il grafico della gerarchia per permettere una comprensione maggiormente intuitiva.

(Le classi contornate da un tratteggio indicano le classi polimorfe astratte, mentre quelle contornate da linea continua quelle concrete)



La realtà proposta dalla nostra applicazione FastSalaries riguarda l’assegnazione degli stipendi distribuiti da una azienda concessionaria ai dipendenti, in base alle vendite effettuate sui mezzi messi a disposizione dall’azienda stessa.

Per la sua realizzazione si è quindi operato provvedendo: alla costruzione di una gerarchia provvista di ereditarietà multipla (a diamante) per la definizione dei differenti tipi di veicoli trattabili, in base alle loro diverse proprietà, e all’implementazione di una classe Lista per immagazzinare ed operare su un contenitore di elementi della gerarchia (e non solo).

Questo contenitore verrà utilizzato con il fine di tenere traccia delle vendite effettuate da un dipendente nel corso della mensilità e per operare sui dipendenti dell’azienda.

**Classi:**

**Primo Livello**

**Mezzo**

E’ la classe base astratta da cui deriveranno poi tutte le tipologie specifiche di mezzi, costituita dagli attributi:

* std::string IdMezzo
* double Potenza
* double Prezzo

Ciò che la rende una classe astratta sono i metodi virtuali puri double getStar() e double getReward() e il distruttore virtuale astratto, il primo metodo è funzionale a ricavare una valutazione ipotetica del mezzo in base alle prestazioni offerte da questo, mentre il secondo restituisce la somma che frutterebbe la sua vendita al dipendente che la effettua in base a specifici bonus derivati dal tipo di vettura venduta. Vengono poi dichiarati e implementati i metodi che permettono la restituzione degli attributi privati della classe:

* std::string getIdMezzo()
* double getPotenza
* double getPrezzo()

**Secondo Livello**

**MezzoCombustione**

E’ una classe virtuale (non implementa il distruttore virtuale puro) derivata virtualmente da Mezzo, estesa con gli attributi:

* double Cilindrata
* static double kPotenzaCombustione
* static double kCilindrata

Ovvero le specifiche generali di un veicolo a combustione e le costanti moltiplicative, marcate static, per pesare le proprietà del mezzo a combustione venduto sul guadagno finale.

La classe implementa i metodi virtuali puri della classe Mezzo come segue:

* double getStar() ritorna la valutazione del mezzo basata sul rapporto cilindrata/potenza, fatta rientrare in precisi intervalli al fine di ottenere un giudizio compreso da 1 a 5.
* double getReward() ritorna il guadagno fruttato dalla vendita del mezzo, ottenuto dalla somma pesata (grazie alle costanti moltiplicative di cui sopra) delle specifiche di potenza e cilindrata.

E il metodo di restituzione dell’attributo privato: double getCilindrata().

**MezzoElettrico**

E’ una classe virtuale (non implementa il distruttore virtuale puro) derivata virtualmente da Mezzo, estesa con gli attributi:

* double capacitaBatteria
* static double kPotenzaElettrico
* static double kBatteria

Segue lo stesso comportamento della classe MezzoCombustione, ma questa volta applicando le proprietà di un mezzo elettrico che abbiamo scelto di contraddistinguere con la capacità della batteria del veicolo ( e la sua relativa costante) e pesando differentemente la potenza.

La classe implementa i metodi virtuali puri della classe Mezzo come segue:

* double getStar() ritorna la valutazione del mezzo basata sul rapporto potenza/batteria, fatta rientrare in precisi intervalli al fine di ottenere un giudizio compreso da 1 a 5.
* double getReward() ritorna il guadagno fruttato dalla vendita del mezzo, ottenuto dalla somma pesata (grazie alle costanti moltiplicative di cui sopra) delle specifiche di potenza e capacitaBatteria.

E il metodo di restituzione dell’attributo privato: double getCapacita().

**Terzo Livello**

**Moto**

E’ una classe concreta derivata da MezzoCombustione che sovrascrive i metodi implementati da quest’ultima aggiungendo la costante moltiplicativa specifica di un veicolo moto, applicata direttamente sul prezzo nel metodo getReward().

**Auto**

E’ una classe concreta derivata da MezzoCombustione che sovrascrive i metodi implementati da quest’ultima aggiungendo la costante moltiplicativa specifica di un veicolo auto, applicata direttamente sul prezzo nel metodo getReward().

**AutoIbrida**

E’ una classe concreta derivata sia da MezzoCombustione che da MezzoElettrico (chiude quindi il diamante) che sovrascrive i metodi implementati da queste classi, aggiungendo la costante moltiplicativa specifica di un veicolo AutoIbrida, applicata direttamente sul prezzo nel metodo getReward() (ottenuto dalla somma dei Reward delle due tipologie di Mezzo) e modificando il metodo getStar() così da ottenere una valutazione figlia della medie delle due calcolate precedentemente

**Monopattino** (Elettrico)

E’ una classe concreta derivata da MezzoElettrico che sovrascrive i metodi implementati da quest’ultima aggiungendo la costante moltiplicativa specifica di un veicolo monopattino, applicata direttamente sul prezzo nel metodo getReward().

.

Oltre alla gerarchia sopra discussa al fine di rendere utilizzabile l’applicazione in una realtà lavorativa, sono state implementate anche le classi

**Lista**

Classe che funge da container degli oggetti utilizzati nel corso dell’utilizzo per permettere un’interazione dinamica dell’utente con l’applicazione che prevede i metodi di:

* InsertFront() -> Inserimento in testa
* InsertBack() -> Inserimento in coda
* Flush() -> Cancellazione di tutti gli oggetti della lista
* getSize() -> ritorna la dimensione attuale della lista

E consente l’utilizzo di Iteratori per scorrere ed operare in un punto qualsiasi della lista.

**Dipendente**

Classe che rappresenta la figura del dipendente della concessionaria, costituita dai seguenti attributi:

* unsigned int id
* Lista<Mezzo\*> vendite
* unsigned int stipendio
* unsigned int sumRating
* unsigned int nvendite

e composta dai metodi di restituzione di questi attributi: unsigned int getId(), unsigned int getStipendio(), unsigned int getSumRating(), unsigned int getNvendite(), Lista<Mezzo\*> getLista().

oltre a questa implementazione “di base” vengono forniti anche i seguenti metodi operativi:

* void addVendita(Mezzo\*) -> aggiunge un mezzo (ovvero una vendita) alla lista di vendite del dipendente
* void rimuoviVendite() -> azzera la lista delle vendite
* double getReward() -> ritorna lo stipendio fruttato dalle vendite presenti al momento nella lista delle vendite del dipendente
* void assegnaRating() -> somma a sumRating i rating dei mezzi venduti, e aggiorna nvendite
* vod assegnaReward -> aggiunge allo stipendio il reward delle vendite e svuota la lista delle vendite, inoltre chiama assegnaRating()
* void azzeraStipendio() -> resetta gli attributi di Dipendente stipendio rating ed nvendite

**Azienda**

Classe che rappresenta la realtà aziendale caratterizzata dall’unico attributo Lista<Dipendente\*> dipendenti e dal corrispondente metodo di restituzione dell’attributo.

Inoltre questa classe si occupa delle operazioni di scrittura/lettura da file costituite dai metodi:

* void aggiungiVendita
* void salvaStipendi
* void azzeraStipendi

Tutti metodi (di cui il comportamento a questo punto risulta deducibile) che aggiornano un file di testo, necessario a tenere traccia e memorizzare permanentemente le informazioni utili all’azienda, calcolate fino a questo punto.

**2. Descrizione del codice polimorfo**

L’uso del polimorfismo all’interno del codice dell’applicazione si nota prevalentemente nell’implementazione e nell’uso dei metodi getReward() e getStar() nella gerarchia di “Mezzo”.

Entrambi i metodi assumono un comportamento differente per ogni oggetto della gerarchia in particolare getReward() viene differenziato in ogni classe concreta dell’ultimo livello per fornire un quadro diversificato e realistico del peso di una vendita sulla base del tipo di mezzo venduto, mentre getStar() utilizza parametri di valutazione differenti in base alle specifiche proprie di ogni differente veicolo.

Nell’applicazione questo si può osservare nella classe “dipendente”, quando al momento della chiamata dei metodi assegnaRating() e assegnaReward() viene passata in rassegna l’intera lista delle vendite corrispondente ai mezzi venduti e utilizzato il late binding per richiamare la funzione corrispondente a ogni differente classe di mezzo presa in esame.

A livello di interfaccia questo meccanismo, avviene nella classe “inseriscivendite”, nel momento in cui viene calcolata la valutazione e solo dopo aver assegnato una vendita ad un determinato dipendente.

**3. Manuale utente della GUI**

All’avvio dell’applicazione l’utente si troverà davanti ad una schermata iniziale dalla quale potrà procedere selezionando una delle tre azioni rese disponibili dall’app, le cui funzionalità risultano molto intuitive dal nome del pulsante.

Inserisci vendite: Permette all’utente di inserire le specifiche di un mezzo venduto e di assegnarle al corrispondente dipendente all’interno dell’azienda individuabile dal suo corrispondente ID, da ricercare all’interno di un menu a tendina che si aprirà al momento del clic sulla sua selezione. Le restanti configurazioni da inserire riguardano le proprietà del mezzo, inseribili sia liberamente, che sotto assistenza di un contatore graduale (i cui incrementi sono calibrati in base alla specifica). Una volta riempito tutti i campi disponibili, e qui si segnala che i campi batteria e cilindrata saranno resi disponibili a seconda del tipo di mezzo scelto (a combustione o elettrico), sarà possibile ottenere una valutazione del mezzo inserito con un giudizio da una a cinque stelle e salvare la vendita, assegnandola effettivamente al dipendente.

Mostra vendite: Presenta una schermata nella quale è possibile visualizzare tutte le vendite inserite dall’operazione precedente, per permettere all’utente di controllare le corrispondenze dei mezzi venduti, per poter passare in rassegna le operazioni effettuate e accorgersi di eventuali errori avvenuti nel momento dell’inserimento delle vendite.

Calcola stipendi: Presenta una schermata nella quale vengono mostrati gli stipendi totali accumulati dalle vendite effettuate dai vari dipendenti, con una loro valutazione a lato, che permette di individuare immediatamente le prestazioni di questi ultimi utili e di fornire un giudizio sul lavoro effettuato sul piano pratico. Una volta presa visione di queste informazioni il processo si può concludere con il comando di cancellazione nella toolbar, che se effettuato, resetta gli stipendi e le valutazioni, rendendo così l’applicazione pronta per un nuovo utilizzo.

**4. Ambiente di sviluppo e di test**

A causa dell’emergenza COVID-19 lo sviluppo e il test sono stati effettuati esclusivamente da casa sulla macchina virtuale fornitaci, di cui sotto si riportano le specifiche

* Sistema operativo: Ubuntu
* Compilatore: gcc 5.4.0
* Libreria Qt 5.4.4

**Istruzioni per la compilazione**

Per compilare il codice è sufficiente lanciare il comando qmake e make all’interno della cartella del file.

**5. Suddivisione del lavoro progettuale e tempistiche**

Il progetto è stato portato avanti parallelamente operando in contemporanea in tutti i vari aspetti dell’applicazione, dalla progettazione all’implementazione, ai fini pratici Davide Pivato si è occupato della realizzazione della parte logica della gerarchia polimorfa e dell’interfaccia grafica delle operazioni di Calcola Stipendi e Mostra Vendite, mentre Andrea De Tomasi si è occupato della realizzazione delle classi Dipendente, Azienda, Lista e della interfaccia grafica dell’operazione “Inserisci Vendite” e della classe PageHandler.

Tempistiche personali:

Concezione del problema -> 4h

Programmazione -> 28h

Debug -> 11 h

Ricerca- > 7h