# Relazione Progetto PAO
# KALK_MENSA

Relazione di: Sharon Della Libera 1121681
Progetto svolto con Giulia Albanello 1125329

2017-2018

## COMPILAZIONE E ESECUZIONE

Per compilare il progetto è necessario eseguire il comando “qmake” nella cartella “kalk_mensa” e poi “make”. Il file KALK_MENSA.pro si trova già all’interno della cartella e non deve essere creato automaticamente con il “qmake -project” in quanto necessita di comandi in più per poter compilare. Per quanto riguarda il file “Use.java” bisogna utilizzare i comandi “javac Use.java” e “java Use” per generare la “Use.class” e vedere le stampe degli esempi nel prompt.

## AMBIENTE DI SVILUPPO

Sistema operativo: Windows 10 Pro
Versione Qt Creator: 4.1.0 Based on Qt 5.7.0
Compilatore: MSVC 2013, 32 bit
Per Java: IDE Eclipse Version Oxygen.2 Release (4.7.2)
Compilatore Java: versione "1.8.0_161"

## MATERIALE CONSEGNATO:

All’interno della cartella kalk_mensa si trovano tutti i file .h .cpp e .pro per l’esecuzione della calcolatrice, il file Use.java e il pdf Relazione.pdf della relazione.

## PRESENTAZIONE

L’idea è di presentare una calcolatrice applicabile ad una mensa in cui vi sono diversi tipi di persone che possono usufruire del servizio e ai quali è associato una riduzione o un esonero in base alla loro tipologia.
La finestra con l’inserimento dei dati dell’utente simula la lettura automatica di un badge; qui ovviamente i dati vengono inseriti manualmente ad ogni cambio utente dopodiché si procede con la calcolatrice vera e propria che dinamicamente mostra le operazioni possibili per ogni persona.

## GERARCHIE

La gerarchia principale è la gerarchia con base Persona, suddivisa in due sottoclassi, Convenzionato e NonConvenzionato, a loro volta suddivise in Studente e Lavoratore per la prima e NonConvenzionatoConBuono per la seconda.
La gerarchia con classe base Cibo serve per completare le funzionalità offerte dalla gerarchia precedente.

- PERSONA: classe base astratta che rappresenta le caratteristiche principali della persona (qui limitate a nome e cognome) con il relativo vector<Cibo*> che contiene i piatti che costituiscono il suo pasto. Offre:
  un metodo double Totale() astratto che verrà implementato nelle sue sottoclassi per calcolare il totale di ogni persona scorrendo il vettore contente i cibi; l’overloading dell’operator + sempre astratto che dovrà aggiungere al vettore il piatto scelto attraverso una push_back; l’overloading dell’operator – che cancella dal vettore un piatto aggiunto scorrendo il vettore e cancellando (stando attenti alle quantità) il piatto che si è scelto di cancellare; il metodo QString stampa_scontrino(double) che ritorna sottoforma di QString i dati della persona (solo nome e cognome); una void reset_v() che svuota il vettore, necessario per non lasciare garbage; un distruttore virtuale che quindi cancella nel modo più corretto possibile un oggetto persona;
  - CONVENZIONATO: classe ancora astratta in quanto non implementa la funzione Totale() per il calcolo del prezzo finale; un convenzionato ha nel suo “badge” (simulato nell’ inserimento utente iniziale) i soldi che ha caricato in esso e che usa nella mensa per prendere il pranzo; implementa l’overloading dell’operatore + controllando che la quantità di ogni piatto scelto sia minore o uguale a 1 poichè i convenzionati hanno diritto ad 1 primo, 1 secondo e 1 contorno e quindi la loro convenzione non permette l’acquisto di più piatti dello stesso tipo. In caso contrario lancia un’eccezione; fornisce poi due metodi per settare i soldi nella carta e ritornarli in quanto il campo dati privato “soldi nella carta” è necessario all’esterno della classe; aggiunge poi al metodi di stampa_scontrino la quantità di soldi nella carta e richiama la stampa scontrino di persona per avere i dati dell’utente;

- ✓ STUDENTE: classe concreta, implementa tutti i metodi astratti e contiene un bool per distinguere una persona con la borsa di studio (quindi con i pasti già pagati) o meno (quindi che usa i soldi nella carta); ha inoltre un valore statico che indica lo sconto che deve essere fatto sul totale in quanto studente (in questo caso -50%); implementa il totale scorrendo il vettore per avere il totale dei piatti presi, applicando lo sconto a lui riservato e se non possiede una borsa sottraendo il totale dalla carta stando attenti a non ottenere un saldo carta negativo; in tal caso viene lanciata un'eccezione; se ha la borsa di studio nello scontrino aggiunge questa informazione e richiama la stampa_scontrino di convenzionato;
- ✓ LAVORATORE: classe concreta, implementa tutti i metodi astratti; contiene un valore statico che indica lo sconto da applicare al totale in quanto è un lavoratore convenzionato con la mensa (lo sconto è del 30%); come in studente implementa l'operator + stando attento a non sforare i soldi nella carta;

  - NONCONVENZIONATO:classe concreta, vi appartengono tutte le persone che usufruiscono della mensa senza avere degli sconti particolari quindi possono prendere i pasti in qualsiasi quantità pagando il prezzo pieno; per questo implementa l'operator + aggiungendo il piatto al vettore senza fare il check sulla quantità;
    - ✓ NONCONVENZIONATOCONBUONO:classe concreta, è un non convenzionato che però ha a disposizione dei buoni da poter usufruire: tale buono può essere usato per prendere 1 primo, 1 secondo e 1 contorno; quindi l'implementazione dell'operator + controlla la quantità di buoni con la quantità di piatti, in modo che se il numero di piatti supera il numero di buoni vengano fatti pagare i piatti in più; anche il non convenzionato con i buoni chiama la funzione stampa_scontrino definita nella base e aggiunge le sue informazioni;
- CIBO:classe astratta, contiene la quantità di quel particolare piatto che la persona prende; è necessario per controllare che i convenzionati non prendano più piatti dello stesso tipo; offre dunque un metodo di set per settare la quantità e l'oveloading dell'operator * che utilizza il metodo precedente per impostare la quantità di cibo; virtualmente dichiara una funzione get_prezzo che dinamicamente ritorna il prezzo di ogni piatto; virtualmente si ha poi il metodo di clonazione necessario per la parte grafica;
  - PRIMO: contiene il suo prezzo fissato, 5.0€;
  - SECONDO: contiene il suo prezzo fissato, 6.0€;
  - CONTORNO: contiene il suo prezzo fissato, 3.0€;
- ECCEZIONE:classe concreta che permette la gestione delle eccezioni nel caso in cui un convenzionato provi a prendere più pasti di quelli che ha a disposizione oppure quando un convenzionato esaurisce i soldi nella carta; offre un metodo QString mostra_errore() che mostra la tipologia di errore sollevata.

## POLIMORFISMO E GUI

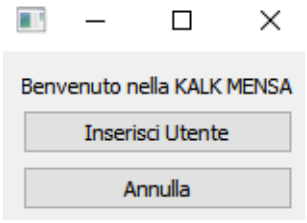
Il polimorfismo è necessario per scegliere dinamicamente (in base alla tipologia della persona) lo sconto da applicare sul totale e per stabilire (sempre dinamicamente) il prezzo dei vari piatti contenuti nel vettore.
Ciò è possibile grazie ai metodi virtuali Totale() nella gerarchia Persona e get_prezzo() nella gerarchia Cibo; un altro metodo virtuale indispensabile è l'operator + che aggiunge al vettore della persona i cibi e in convenzionato controlla che vi sia un solo piatto per ogni tipo mentre in non convenzionato esegue la push_back senza problemi.
Anche stampa_scontrino(double) è virtuale in quanto stampa le informazioni della persona dinamicamente.

Nella gui ad ogni cambio utente viene costruito un puntatore polimorfo di tipo statico Persona che punta al nuovo utente inserito attraverso la new (es: Persona* person= new Studente("sha","dl",45,false) ) sfruttando così il polimorfismo. In questo modo ogni metodo virtuale invocato con "person->metodo_virtuale" invoca la funzione corretta a run-time. Ogni volta poi che si effettua il cambio utente ci si occupa della cancellazione della persona per non lasciare garbage nello heap.

Succede la stessa cosa per quanto riguarda la dichiarazione di un cibo: al puntatore Cibo* viene assegnata una new ad ogni click su una tipologia di piatto (Cibo* c=new Primo() ) che verrà usato in base alle operazioni scelte (come la +, che inserisce nel vector<Cibo*> la copia di un puntatore c).
**Attenzione:** si inserisce una copia per evitare la condivisione di memoria: inserendo l'oggetto c senza copia sia c che l'ultimo puntatore di vector puntano allo stesso oggetto e quando, per non lasciare garbage, elimino c per riassegnarlo, lo eliminerei dal vettore, cose che non va bene.

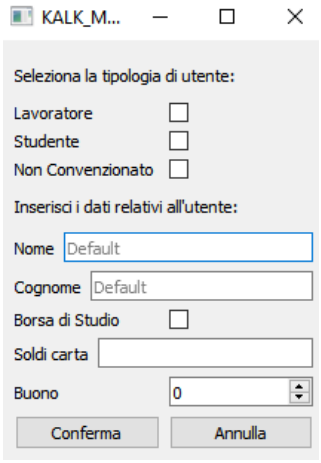
## MANUALE GUI

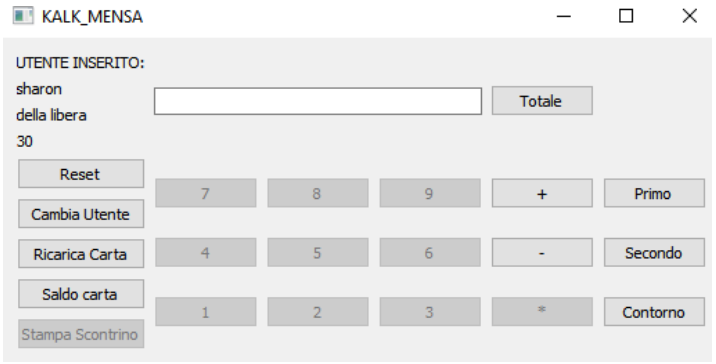
Chi utilizza la calcolatrice deve essere a conoscenza dei diversi tipi di persona esistente per inserire correttamente i dati nella finestra "inserisci utente". L'uso della calcolatrice vera e propria invece è agevolato dall'oscuramento dei pulsanti non consentiti per quell'utente. Infatti, ad esempio, se si sceglie un convenzionato il tastierino con i numeri che consente l'operazione * viene oscurato per impedire l'acquisto di più piatti dello stesso tipo.
L'interfaccia è molto semplice e schematica, ma molto facile da usare.




**Finestra principale:**
Consiste semplicemente in due pulsanti: Inserisci Utente che ci permette di accedere alla seconda finestra per scegliere l'utente e Annulla che esce totalmente dall'esecuzione della calcolatrice.

**Scelta dell'utente**
La scelta principale è tra lavoratore,studente e non convenzionato.
Per ognuno, se selezionato, vengono nascosti i pulsanti non necessari per la creazione del nuovo utente.




**Calcolatrice**
Nella figura è stato inserito uno studente le cui informazioni vengono riportati in alto a sinistra.
Da questa finestra possiamo eseguire le operazioni e mostrare il totale cliccando sul pulsante Totale e stampare lo scontrino a fine operazioni cliccando su Stampa Scontrino (ora oscurato).
Le operazioni consentite sono: la + che aggiunge il pasto e la – che sottrae il pasto in caso di errore da parte di chi sta usando la calcolatrice.

Possiamo cambiare l'utente una volta completate le operazioni, ricaricarne la carta, mostrare il saldo della carta e resettare le operazioni effettuate fino a quel momento utilizzando i pulsanti sulla sinistra.

**NB: Dopo aver ottenuto il totale, selezionare Stampa Scontrino per vedere la finestra che simula lo scontrino.**

**NB: L'operatore * (nel caso di non convenzionato) funziona solo se si clicca prima il tipo di cibo e poi il numero che corrisponde alla quantità presa dalla persona. Altrimenti il numero, se cliccato prima, non viene preso in considerazione. Esempio:**
**primo*3 -> ottengo il risultato corretto;**
**3*primo -> invece non considera né il numero né l'operatore *.**

## ORE IMPIEGATE E SUDDIVISIONE COMPITI

Si è scelto di suddividere così il lavoro, confrontandoci spesso per controllare eventuali errori:
Implementazione modello: assegnata a Sharon Della Libera;
Implementazione grafica: assegnata a Giulia Albanello;

Nel mio caso le ore si sono suddivise così:
Analisi e stesura della struttura generale delle gerarchie: 6 ore.
Implementazione della logica: 26 ore.
Implementazione di alcuni metodi della grafica: 6ore.
Test finale, codice java e prove di compilazione: 7 ore.
Relazione: 3 ore.

Il totale è dunque di 48 ore.