POLITECNICO DI TORINO

Laurea di 1° livello in Ingegneria Gestionale

Classe L-8 Ingegneria dell'Informazione

IL TEATRINO - SPAZIO PER CORSI

RELATORE
Prof. Fulvio Corno

CANDIDATO
Andrea Mascarello

Anno Accademico
2018/2019

INDICE

1. PROPOSTA DI PROGETTO

Studente proponente

s233799 Mascarello Andrea

Titolo della proposta

Gestione dati clienti e ottimizzazione corsi de "Il Teatrino"

Descrizione del problema proposto

Ho contattato la titolare de "Il Teatrino", attività in proprio che offre corsi di teatro, che mi ha fornito un database relativo ai clienti. L' applicazione dovrà andare a risolvere i problemi della titolare, quali la difficoltà nel trovare informazioni relative al singolo cliente, la difficoltà nel calcolare il fatturato (mensile ed annuale) e la difficoltà nel modificare i dati relativi ai clienti. In secondo luogo con i dati forniti relativi alle spese (luce, affitto, dipendenti) l'applicazione andrà a calcolare la strategia che permette di massimizzare i guadagni creando un orario settimanale per i corsi da svolgere.

Descrizione della rilevanza gestionale del problema

Dal punto di vista gestionale, l'applicazione porterà a risparmiare ore di lavoro su fogli Excel permettendo all'utente di trovare (e/o modificare) dati relativi ai clienti o ai singoli corsi. Inoltre permetterà anche di calcolare il fatturato (mensile e/o annuale) totale o relativo ai singoli corsi (permettendo all'utente di conoscere quali sono i corsi più proficui) ed il guadagno orario. L'applicazione sarà anche in grado di calcolare quali sono i clienti più presenti dato lo storico dei corsi, in tal modo si potrà capire quali sono i "migliori". Infine ha grande rilevanza gestionale l'ottimizzazione dei corsi che restituirà all'utente una strategia ottima per quanto riguarda gli orari e le date in cui svolgere i corsi e la necessità.

Descrizione dei data-set per la valutazione

I dati sono estrapolati da tabelle Excel e inseriti in un database relazionale comprendente 6 tabelle:

- clienti: ID cliente, nominativi cliente(modificati per privacy), ID corso frequentato, contatti , compenso pagato, compenso da pagare

- corsi : ID corso , nome corso, costo, ID dipendente presente, ID secondo dipendente, giorno della settimana, orario di inizio, orario di fine ,tipologia

- dipendenti : ID , nome dipendente(modificato per privacy), compenso orario, giorno libero

- eventi : nome, data, entrata

- lezioni : id lezione, id corso, data, presenze

- iscrizioni : id corso, id cliente

Dati aggiuntivi sono : costo dell' affitto dei locali, spese varie.

Descrizione preliminare degli algoritmi coinvolti

In un primo momento gli algoritmi coinvolti saranno quasi semplicemente utili per interrogare il database, al fine di restituire all'utente i dati relativi ai singoli clienti, ai corsi, ai dipendenti, al fatturato(dei singoli corsi o mensile).In secondo luogo l' applicazione andrà a provare una serie di combinazioni relative ai corsi, sfruttando un algoritmo ricorsivo per ottenere la strategia che permetterà di massimizzare i profitti offrendo un orario settimanale dei corsi. Questa seconda parte terrà conto di una serie di fattori quali l'affitto dei locali, le spese per dipendenti e luce, il periodo dell' anno in cui si svolge un corso, il giorno e l'orario in cui si svolge.

Descrizione preliminare delle funzionalità previste per l'applicazione software

L'applicazione sarà dotata di diverse interfacce grafiche attraverso le quali sarà possibile utilizzare tutte le funzioni dell'applicazione.
L'idea principale era quella di separare le sei macro funzioni dell'applicazione. Attraverso un menù principale sarà possibile decidere se accedere all'area di gestione e modifica dati, a quella di calcolo dei valori economici o passare al calcolo della strategia ottimale.

- **Gestione e modifica dati:**

L' utente (che ha specificatamente richiesto queste funzioni per alleggerire il proprio lavoro) potrà svolgere differenti attività, tra le quali:

1. Inserimento (e/o eliminazione) di tutti i dati relativi ad un nuovo (o vecchio) cliente e/o corso, registrazione pagamenti da parte dei clienti, ricerca di tutti i dati relativi ad un singolo cliente.
2. Ricerca di tutti i dati relativi ad un corso, aggiunta ed eliminazione corsi, con aggiunta di lezioni e aggiornamento dei compensi dovuti dai clienti.
3. Visualizzazione attraverso grafici dei compensi guadagnati mensilmente o riguardo i singoli corsi.
4. Dati relativi ai dipendenti con possibile aggiunta o eliminazione di uno di essi, assieme alla visualizzazione con grafico a barre dei compensi pagati ad ogni singolo dipendente.
5. Calendario degli eventi passati e futuri, con visualizzazione di un grafico a linea per le entrate registrate.

- **Calcolo strategia ottimale**

In questa sezione dell'applicazione l'utente sarà in grado di preparare un piano per i periodi futuri. Prima di procedere con l'ottimizzazione egli dovrà inserire le ore di disponibilità (dei locali o dei clienti).

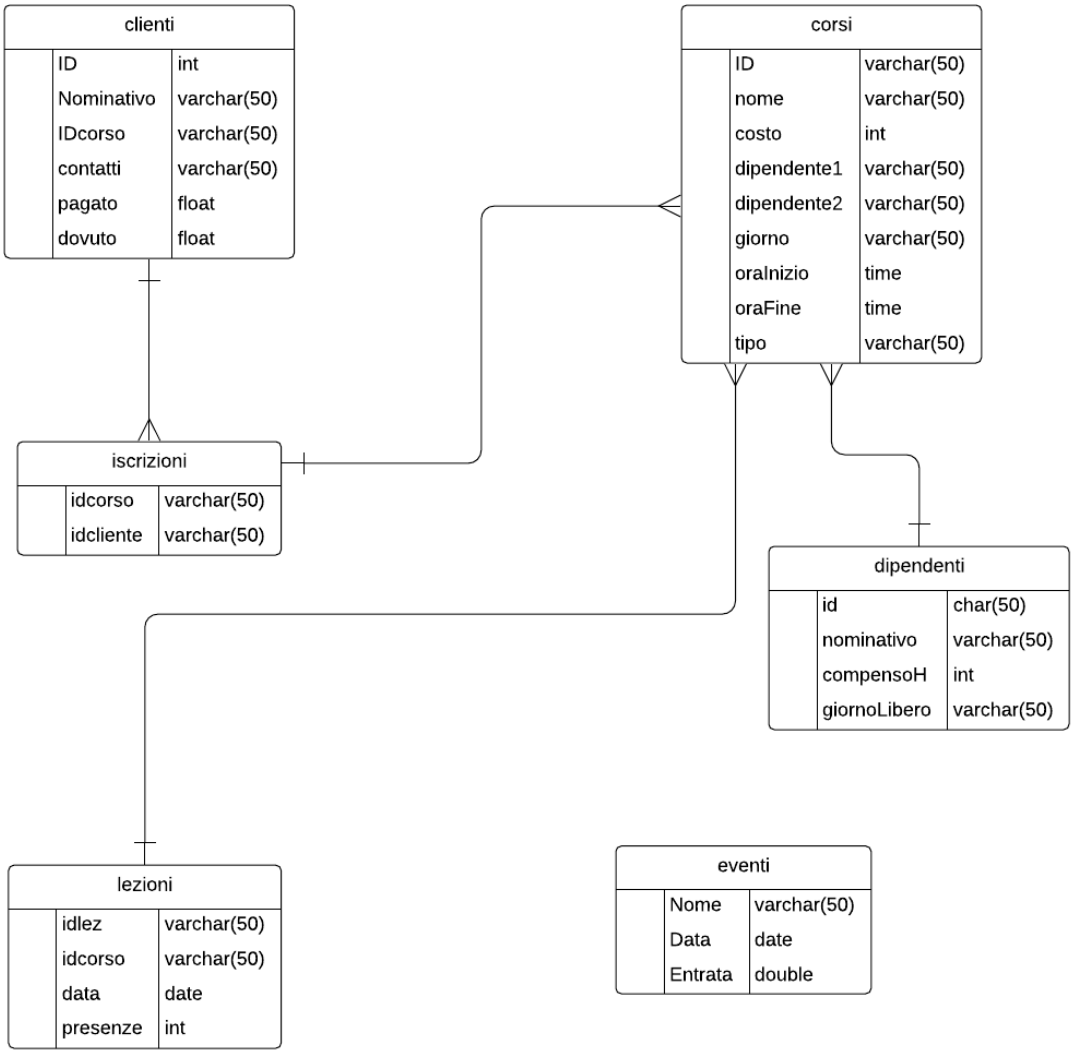
A questo punto l’applicazione troverà le possibili combinazioni settimanali dei corsi (considerando disponibilità locali e giorni liberi dei dipendenti), restituendo in output quella che garantisce un profitto maggiore.

2. DESCRIZIONE DEL PROBLEMA

La titolare de “Il Teatrino” aveva difficoltà a gestire i dati dei numerosi clienti, in quanto Excel presenta grossi limiti rispetto ad un applicativo gestionale.
In particolare la privacy dei clienti non era garantita in quanto al momento del pagamento il cliente poteva vedere sul foglio di lavoro non solo i propri dati ma anche quelli relativi ai suoi compagni di corso.
Con Excel risultava anche molto una panoramica dei guadagni e addirittura era impossibile una visualizzazione grafica.
La gestione dei corsi e dei dipendenti era praticamente nulla in quanto impossibile.
L’aggiunta di corsi, clienti e dipendenti risultava lunga ed andava a modificare tutta la struttura del foglio di lavoro.
Gli eventi erano riportati senza una logica e non era possibile calcolare il guadagno totale ottenuto da essi.

Per quanto riguarda la parte relativa all’ottimizzazione, nasce dalla mancanza di un pensiero strategico. Il Teatrino offre corsi a numerosi clienti, portando buoni guadagni alla titolare, ma con una strategia questi potrebbero diventare ottimi. I corsi negli ultimi due anni venivano inseriti quasi a caso nei giorni della settimana, seguendo le disponibilità dei dipendenti e dei clienti.
Con questo algoritmo però, la titolare sarà in grado non solo di massimizzare i profitti, ma anche di decidere quando lavorare (poiché l’algoritmo trova una sequenza di corsi con le ore di disponibilità che vengono inserite).
Se perciò all’inizio della settimana si trovasse in difficoltà potrebbe capire se possibile svolgere tutti i corsi nonostante gli impegni extra lavorativi, sapendo addirittura il profitto massimo stimato per quella settimana (stimato perché nel calcolo del profitto si considera una media di presenze ai corsi).

3. DESCRIZIONE DEL DATA-SET



I dati sono stati estrapolati da fogli Excel contenenti dati sui clienti, sulle lezioni, sugli eventi esterni ai corsi e sui dipendenti.

Da questi dati ho creato un database relazionale contenente 6 tabelle : clienti, corsi, dipendenti, eventi, iscrizioni e lezioni.

Clienti, contiene informazioni su coloro che partecipano ai corsi :

- ID : identifica univocamente il cliente (INT)
- Nominativo : il nome del cliente (VARCHAR di massimo 50 caratteri)
- ID corso : codice del corso a cui è iscritto il cliente (VARCHAR di massimo 50 caratteri)

- Contatti : email del cliente (VARCHAR di massimo 50 caratteri, può essere NULL)
- Pagato : compenso pagato dal cliente (FLOAT)
- Dovuto : compenso dovuto dal cliente (FLOAT)

Corsi, contiene informazioni sui singoli corsi :

- ID : identifica univocamente il corso (VARCHAR di massimo 50 caratteri)
- Nome : nome del corso (VARCHAR di massimo 50 caratteri)
- Costo : costo di una singola lezione (INT)
- IDdipendente1 : Identifica il dipendente principale della lezione (VARCHAR di massimo 50 caratteri)
- IDdipendente2 : Identifica il dipendente secondario della lezione (VARCHAR di massimo 50 caratteri)
- Giorno : giorno in cui si svolge la lezione (VARCHAR di massimo 50 caratteri)
- OraInizio : ora di inizio del corso (TIME)
- OraFine : ora di fine del corso (TIME)
- Tipo : identifica il tipo di pagamento (VARCHAR di massimo 50 caratteri), può essere TRIM (trimestrale) o LEZ (lezione).

Dipendenti, contiene informazioni sui dipendenti :

- ID : identifica univocamente il dipendente (VARCHAR di massimo 50 caratteri)
- Nominativo : nome del dipendente (VARCHAR di massimo 50 caratteri)
- compensoH : compenso orario del dipendente (INT)
- giornoLibero : giorno libero del dipendente (VARCHAR di massimo 50 caratteri)

Iscrizioni, permette di sapere a quale corso è iscritto ogni cliente:

- idCorso : identificativo del corso (VARCHAR di massimo 50 caratteri)
- idCliente : identificativo del cliente (VARCHAR di massimo 50 caratteri)

Lezioni, tiene traccia di tutte le lezioni avvenute:

- idLez : identifica univocamente la lezione (VARCHAR di massimo 50 caratteri)
- idCorso : identifica il corso della lezione (VARCHAR di massimo 50 caratteri)
- data : quando si è svolta la lezione (DATE)
- presenze : identifica quanti studenti hanno partecipato alla lezione (INT)

Eventi :

- Nome : nome dell'evento (VARCHAR di massimo 50 caratteri)
- Data : data in cui si è tenuto l'evento (DATE)
- Entrate : soldi guadagnati dall'evento (DOUBLE)

4. DESCRIZIONE DELLE STRUTTURE DATI E DEGLI ALGORITMI

L'applicazione è stata sviluppata in linguaggio Java, con il supporto delle interfacce JavaFX, sono implementati il pattern MVC (Model – View – Controller) ed il pattern DAO (Data Access Object).
L'applicazione si divide in tre package :

- teatrino :

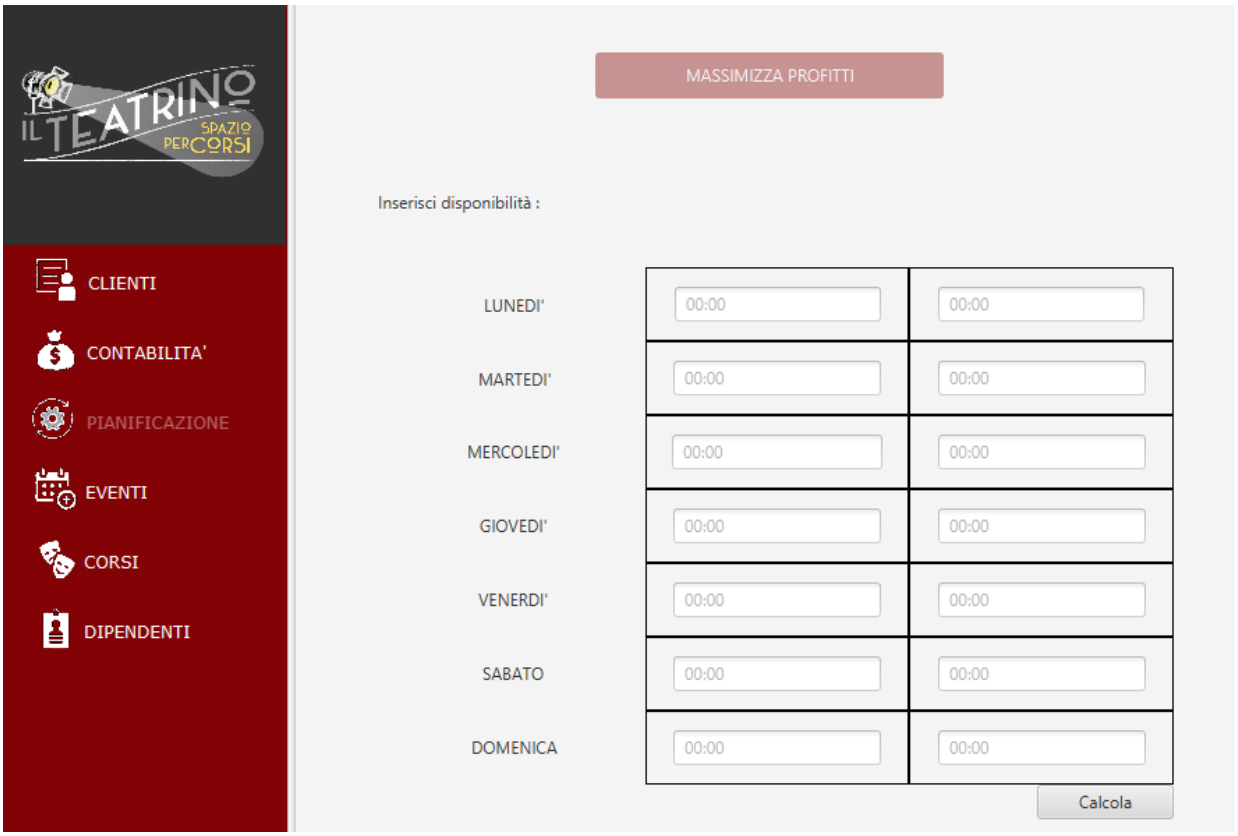
Contiene i vari fogli FXML relativi alle interfacce grafiche ed i controller di questi ultimi.
Vi sono 22 fogli FXML e i 22 relativi Controller, inoltre sono presenti una classe Main ed un foglio di stile css (non modificato, le modifiche sono state fatte direttamente negli oggetti in SceneBuilder)

- model :

Contiene la classe Model, nella quale si trova l'intera logica applicativa del software. Le altre classi sono relative agli oggetti utilizzati nello sviluppo : Cliente, Corso, Dipendente, ClienteHidden (utilizzata per la TableView), Lezione, Evento, Giorno (particolarmente rilevante per l'algoritmo ricorsivo).

- db :

Contiene le classi ConnectDB (che permette il collegamento tra Eclipse e HeidiSQL) e TeatrinoDAO (contenente tutte le istruzioni in linguaggio MySQL che permettono di ricavare valori dal database)
Tutta la parte relativa alla gestione, aggiunta, modifica, visualizzazione, eliminazione dati non contiene parti rilevanti a livello algoritmico. Mi concentrerò quindi sulla spiegazione dell'algoritmo che ha richiesto più tempo e ragionamento : il ricorsivo utilizzato per l'ottimizzazione dei corsi.
Con un approccio top-down, andiamo a visualizzare l'algoritmo.
L'utente attraverso la schermata sottostante inserisce le disponibilità di locali e/o clienti.



Attraverso questo codice presente nel Controller dell'ottimizzazione, i valori inseriti in input vengono passati al model che tramite il metodo trovaSequenza() andrà a calcolare la lista di corsi da restituire in output.

Se la sequenza conterrà tutti i corsi verrà passata ad un controller per la visualizzazione e attraverso un interfaccia verrà mostrata all'utente.

Il resto del controller contiene gestione di errori input ed eccezioni.

```
@FXML
void ricorsivo(ActionEvent event) {

    String output = "";

    try {
        LocalTime lunediI = LocalTime.parse(txtInizioL.getText());
        LocalTime lunediF = LocalTime.parse(txtFineL.getText());
        LocalTime martediI = LocalTime.parse(txtInizioMa.getText());
        LocalTime martediF = LocalTime.parse(txtFineMa.getText());
        LocalTime mercolediI = LocalTime.parse(txtInizioMe.getText());
        LocalTime mercolediF = LocalTime.parse(txtFineMe.getText());
        LocalTime giovediI = LocalTime.parse(txtInizioG.getText());
        LocalTime giovediF = LocalTime.parse(txtFineG.getText());
        LocalTime venerdiI = LocalTime.parse(txtInizioV.getText());
        LocalTime venerdiF = LocalTime.parse(txtFineV.getText());
        LocalTime sabatoI = LocalTime.parse(txtInizioS.getText());
        LocalTime sabatoF = LocalTime.parse(txtFineS.getText());
        LocalTime domenicaI = LocalTime.parse(txtInizioD.getText());
        LocalTime domenicaF = LocalTime.parse(txtFineD.getText());

        if(lunediI.compareTo(LocalTime.of(7, 00)) >= 0 && (martediI.compareTo(LocalTime.of(7, 00)) >= 0) &&
                mercolediI.compareTo(LocalTime.of(7, 00)) >= 0 &&
                mercolediI.compareTo(LocalTime.of(7, 00)) >= 0 && giovediI.compareTo(LocalTime.of(7, 00)) >= 0
                && venerdiI.compareTo(LocalTime.of(7, 00)) >= 0 && sabatoI.compareTo(LocalTime.of(7, 00)) >= 0
                && domenicaI.compareTo(LocalTime.of(7, 00)) >= 0) {


        Duration orariTotali = Duration.between(lunediI, lunediF).plus(Duration.between(martediI, martediF)).
                plus(Duration.between(mercolediI, mercolediF)).plus(
        Duration.between(giovediI, giovediF)).plus(Duration.between(venerdiI, venerdiF))
                .plus(Duration.between(sabatoI, sabatoF)).plus(Duration.between(domenicaI,domenicaF));

        Duration orariNecessari = Duration.between(LocalTime.of(17, 00), LocalTime.of(17, 00));

        for(Corso corso : model.getCorsiMap().values()) {
```

```
            System.out.println(Duration.between(corso.getOraInizio(), corso.getOraFine()).toMinutes());
            orariNecessari = orariNecessari.plusMinutes(Duration.between(corso.getOraInizio(),
                    corso.getOraFine()).toMinutes());

        }
        System.out.println(orariTotali.toMinutes());
        System.out.println(orariNecessari.toMinutes());

        if(orariTotali.compareTo(orariNecessari) >= 0) {

        if(lunediI.compareTo(lunediF) <= 0 && martediI.compareTo(martediF) <= 0 && mercolediI.compareTo(mercolediF)
                <= 0 && giovediI.compareTo(giovediF) <= 0
                && venerdiI.compareTo(venerdiF) <= 0 && sabatoI.compareTo(sabatoF) <= 0 &&
                domenicaI.compareTo(domenicaF) <= 0) {

            List<Corso> corsi = model.trovaSequenza(lunediI,lunediF,martediI,martediF,mercolediI,mercolediF,
                    giovediI,giovediF,venerdiI,venerdiF,
                sabatoI,sabatoF,domenicaI,domenicaF);

            for(Corso corso : corsi) {

                System.out.println(corso.toStringRicorsiva());

            }

            System.out.println("Con profitto : "+model.getProfitto(corsi));

            if(corsi.size() > 0) {

            try {

                FXMLLoader loader = new FXMLLoader(getClass().getResource("VisualizzaSequenza.fxml"));
                BorderPane root = (BorderPane) loader.load();
                Scene scene = new Scene(root);
                Stage s = new Stage();
                scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());

                VisualizzaSequenzaController controller = loader.getController();
                controller.setModel(model, s, corsi);

                s.setScene(scene);
                //s.setAlwaysOnTop(true);
                s.show();

                } catch (Exception e) {
```

Il metodo trovaSequenza() crea una nuova lista best e una lista parziale. Va inoltre a ripulire la lista di giorni settimana per poi riaggiungere ad essa tutti i valori in input inseriti. Dopo ciò fa partire la ricorsione e alla fine di essa restituisce la lista best.

```
    public List<Corso> trovaSequenza(LocalTime lunediI, LocalTime lunediF, LocalTime martediI, LocalTime martediF,
            LocalTime mercolediI, LocalTime mercolediF, LocalTime giovediI, LocalTime giovediF, LocalTime venerdiI,
            LocalTime venerdiF, LocalTime sabatoI, LocalTime sabatoF, LocalTime domenicaI, LocalTime domenicaF) {
        // TODO Auto-generated method stub
        best = new ArrayList<Corso>();
        List<Corso> partial = new ArrayList<Corso>();
        settimana.clear();

        settimana.add(new Giorno("Lunedì", lunediI, lunediF));
        settimana.add(new Giorno("Martedì", martediI, martediF));
        settimana.add(new Giorno("Mercoledì", mercolediI, mercolediF));
        settimana.add(new Giorno("Giovedì", giovediI, giovediF));
        settimana.add(new Giorno("Venerdì", venerdiI, venerdiF));
        settimana.add(new Giorno("Sabato", sabatoI, sabatoF));
        settimana.add(new Giorno("Domenica", domenicaI, domenicaF));

        recursion(partial,0,settimana);

        return best;
    }
```

Qui viene mostrata la classe Giorno, fondamentale per la ricorsione, essa contiene infatti ore di inizio e fine, permettendo di conoscere

quante ore rimangono disponibili nel giorno e di stilare quindi un calendario settimanale senza sovrapporre corsi.
Contiene anche una lista di corsi, poiché nei livelli successivi al primo della ricorsione, non è possibile reinserire un corso in un giorno in cui è già stato.

```java
package it.polito.tdp.itunes.model;

import java.time.Duration;

public class Giorno {

    String nome;
    LocalTime oraI;
    LocalTime oraF;
    long disponibilita;
    List<String> corsi = new ArrayList<String>();

    public Giorno(String nome, LocalTime oraI, LocalTime oraF) {
        super();
        this.nome = nome;
        this.oraI = oraI;
        this.oraF = oraF;
        disponibilita = Duration.between(oraI, oraF).toMinutes();
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public LocalTime getOraI() {
        return oraI;
    }
    public void setOraI(TemporalAmount oraI) {
        this.oraI = this.oraI.plus(oraI);
    }
    public LocalTime getOraF() {
        return oraF;
    }
    public void setOraF(LocalTime oraF) {
        this.oraF = oraF;
    }

    public void disponibilita(long amount) {
```

Qui il metodo ricorsivo utilizzato, particolare rilevanza hanno il metodo per identificare il giorno spiegato in seguito, il metodo getProfitto() spiegato anche esso in seguito. Come si può notare il giorno viene salvato nel corso, per evitare che venga riassegnato ad altri e l'ora di inizio di ogni giorno viene modificata ad ogni aggiunta, permettendo di inserire un corso quando quello precedente finisce.

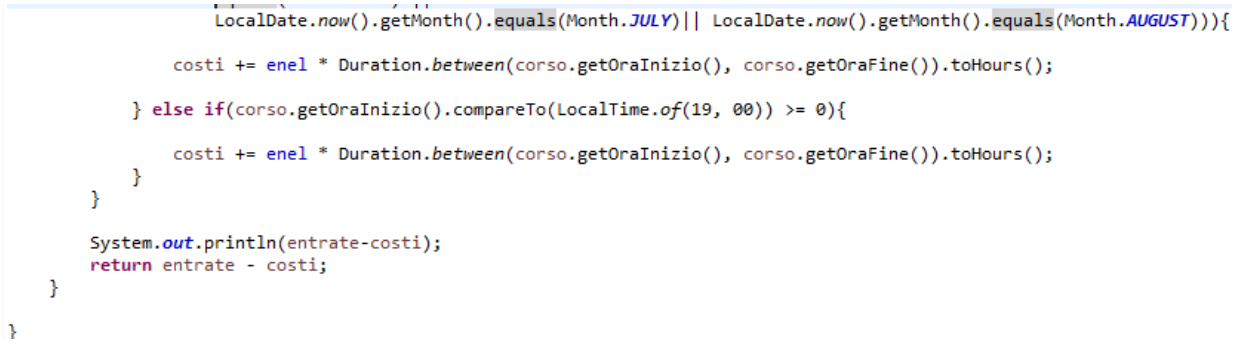
```
    private void recursion(List<Corso> partial,int L ,List<Giorno> sett) {
        // TODO Auto-generated method stub
        System.out.println("PARTITA RICORSIONE");

        if (L == corsiMap.values().size() && getProfitto(partial) > getProfitto(best)) {

            System.out.println("NEW BEST");
            best = new ArrayList<Corso>(partial);
            System.out.println(getProfitto(best));
            System.out.println("ESCO LIVELLO MAX");
            return;
        }

        if (L == corsiMap.values().size()) {

            System.out.println("ESCO");
            return;
        }

        for (Corso corso : corsiMap.values()) {

            if(!partial.contains(corso)) {
            Giorno gg = cercaGiorno(corso,sett,partial);
            if(gg != null) {
            Corso c = new Corso(corso.getId(), corso.getNome(), corso.getCosto(), corso.getDip1(),
                    corso.getDip2(), gg.nome, gg.oraI, gg.oraI.plus(Duration.between(corso.getOraInizio(),
                            corso.getOraFine())),
                    corso.getTipo());
            partial.add(c);
            c.setGg(gg);
            gg.addCorsi(c.getId());
            System.out.println("AGGIUNGO : "+new Corso(corso.getId(), corso.getNome(), corso.getCosto(), corso.getDip1(
                    corso.getDip2(), gg.nome, gg.oraI, gg.oraI.plus(Duration.between(corso.getOraInizio(), corso.getOra
                    corso.getTipo()).toStringRicorsiva());
            if(gg.getOraI().plus(Duration.between(corso.getOraInizio(), corso.getOraFine())).compareTo(gg.oraF) <= 0 )
            gg.setOraI(Duration.between(corso.getOraInizio(), corso.getOraFine()));
            }


          gg.disponibilita(Duration.between(corso.getOraInizio(), corso.getOraFine()).toMinutes());
          for (Corso corso2 : partial) {
              System.out.println(corso2.toStringRicorsiva());
          }

          recursion(partial,L+1,sett);
          System.out.println("RICORRO AL LIVELLO : "+(L+1));
          System.out.println("RIMUOVO : "+partial.get(partial.size()-1));
          partial.get(partial.size()-1).getGg().setOraI(Duration.between(partial.get(partial.size()-1).getOraFine
                  partial.get(partial.size()-1).getOraInizio()));
          partial.remove(partial.size()-1);
          }
          }
      }
  }
```

Questo il metodo per trovare il giorno in cui inserire il corso. Con il ciclo sul parziale si evita che al corso venga assegnato un'ora già occupata (in realtà questo avviene settando l'ora, ma quando si passa dalle 22 alle 00 per l'algoritmo spiegato in precedenza, l'ora di inizio non viene settata. Questo ciclo è utile in questa situazione).

```
private Giorno cercaGiorno(Corso corso, List<Giorno> sett, List<Corso> partial) {
    // TODO Auto-generated method stub
    for(Giorno giorno : sett) {

        boolean err = false;

        for(Corso c : partial) {


            if(c.getGg().nome.equals(giorno.getNome()) && c.getOraInizio().equals(giorno.getOraI()))
                err = true;

        }

        if(err == false) {

        if(!giorno.corsi.contains(corso.getId())) {
        if(!giorno.getOraI().equals(giorno.getOraF()) && giorno.getOraI().plus(Duration.between(corso.getOraInizio(
                , corso.getOraFine())).compareTo(giorno.oraF) <= 0  &&
                !dipendentiMap.get(corso.getDip1()).getGiorno().equals(giorno.nome) && giorno.getOraI()
                .compareTo(LocalTime.of(7, 00)) >= 0 &&
                giorno.getOraI().compareTo(LocalTime.of(23, 00)) <= 0) {
            if(dipendentiMap.get(corso.getDip2()) != null) {
                if(!dipendentiMap.get(corso.getDip2()).getGiorno().equals(giorno.nome)) {
                    return giorno;
            }
            } else {
                return giorno;
            }
        }

    }
    }
    }
    return null;
}
```

Il metodo getProfitto() calcola l'entrata prevista con quella sequenza di corsi. Considera come entrate il costo del corso moltiplicato per la media delle presenze e come costi il pagamento ai singoli dipendenti ("Giovanna Candi" è la titolare, considerata qui dipendente, perciò il suo pagamento non è costo ma dividendo), il pagamento del parcheggio, che varia nei giorni di mercato e il costo della luce (si considera una stima di quando cala il sole in base al periodo dell'anno in cui ci si trova).

```
public double getProfitto(List<Corso> partial) {
    // TODO Auto-generated method stub
    double entrate = 0.0;
    double costi = 0.0;
    costi += affitto/4;

    for(Corso corso : partial) {

        entrate += corso.getCosto() * dao.mediaPresenze(corso);
        if(!corso.getDip1().equals("Giovanna Candi"))
            costi += dipendentiMap.get(corso.getDip1()).getCompenso()*(Duration.between(corso.getOraInizio(),
                    corso.getOraFine()).toHours());
        if(dipendentiMap.get(corso.getDip2()) != null) {
            costi += dipendentiMap.get(corso.getDip2()).getCompenso()*(Duration.between(corso.getOraInizio(),
                    corso.getOraFine()).toHours());
        }
        if(corso.getGiorno().equals("Martedì") || corso.getGiorno().equals("Giovedì") || corso.getGiorno().equals
                ("Sabato")) {
                costi += parcheggioMercato * Duration.between(corso.getOraInizio(), corso.getOraFine()).toHours();
        } else {

                costi += parcheggio  * Duration.between(corso.getOraInizio(), corso.getOraFine()).toHours();
        }
        if(corso.getOraInizio().compareTo(LocalTime.of(17, 00)) >= 0 && (LocalDate.now().getMonth().equals
                (Month.DECEMBER) ||
                LocalDate.now().getMonth().equals(Month.JANUARY)|| LocalDate.now().getMonth().equals(Month.FEBRUARY

            costi += enel * Duration.between(corso.getOraInizio(), corso.getOraFine()).toHours();

        } else if(corso.getOraInizio().compareTo(LocalTime.of(19, 00)) >= 0 && (LocalDate.now().getMonth().
                equals(Month.MARCH) ||
                LocalDate.now().getMonth().equals(Month.APRIL)|| LocalDate.now().getMonth().equals(Month.MAY))) {

            costi += enel * Duration.between(corso.getOraInizio(), corso.getOraFine()).toHours();

        } else if(corso.getOraInizio().compareTo(LocalTime.of(21, 30)) >= 0 && (LocalDate.now().getMonth().
                equals(Month.JUNE) ||
```

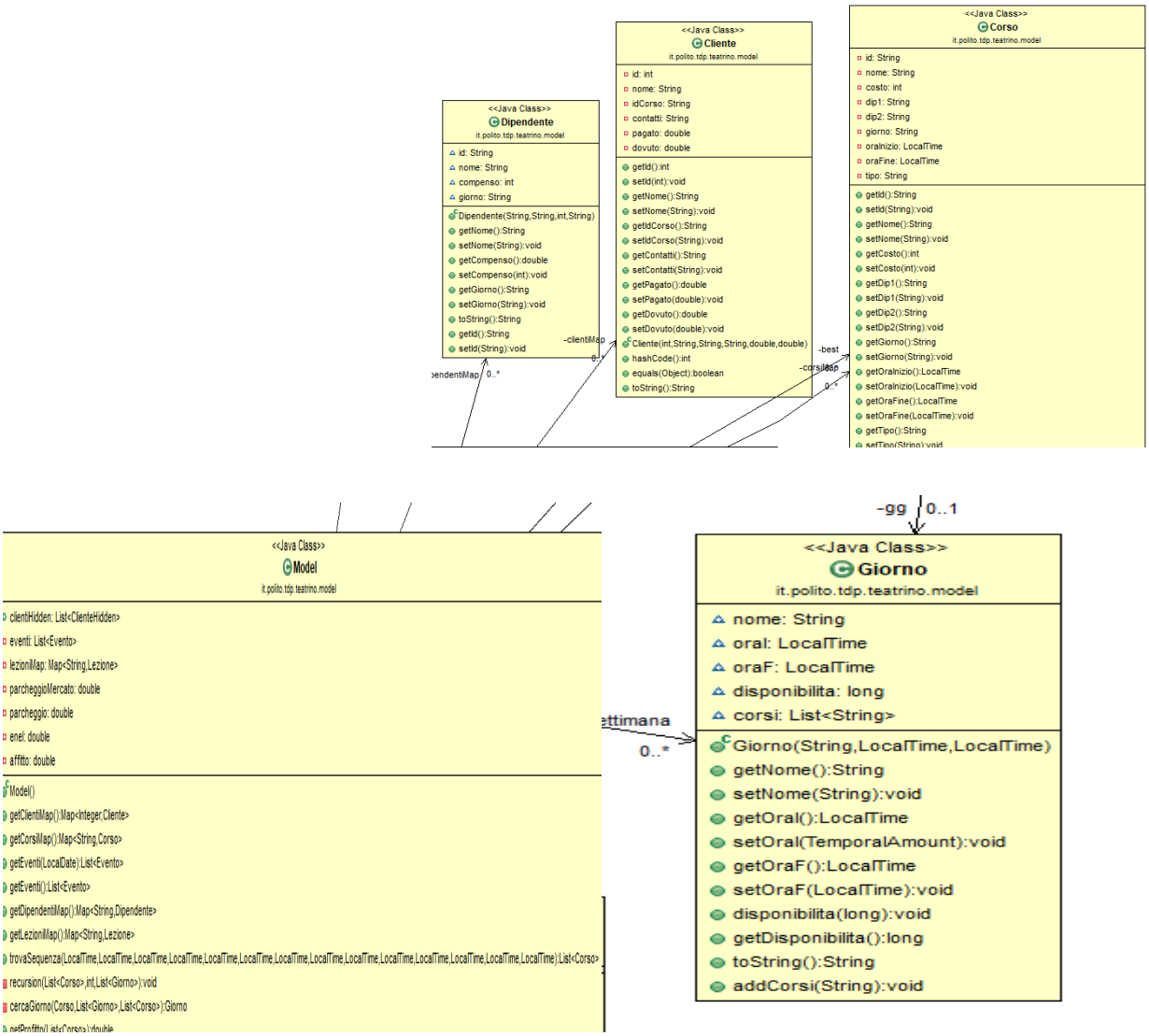


Si tenga conto del fatto che :

- l'utente non può inserire come input orari minori delle 7 AM o superiori delle 11 PM
- l'utente non può inserire ore fine minori delle ore inizio
- gli orari sono da inserire nel formato hh:mm
- devono esserci abbastanza ore disponibili per far partire l'algoritmo

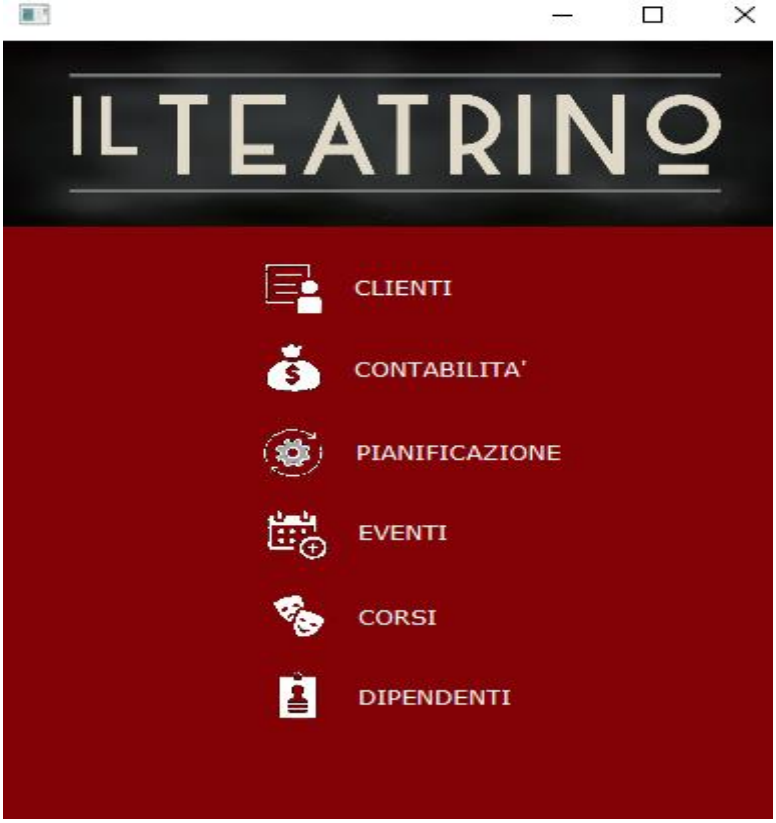
5. DIAGRAMMA DELLE CLASSI

Per una visione più ampia consultare il diagramma nella cartella documenti del progetto.

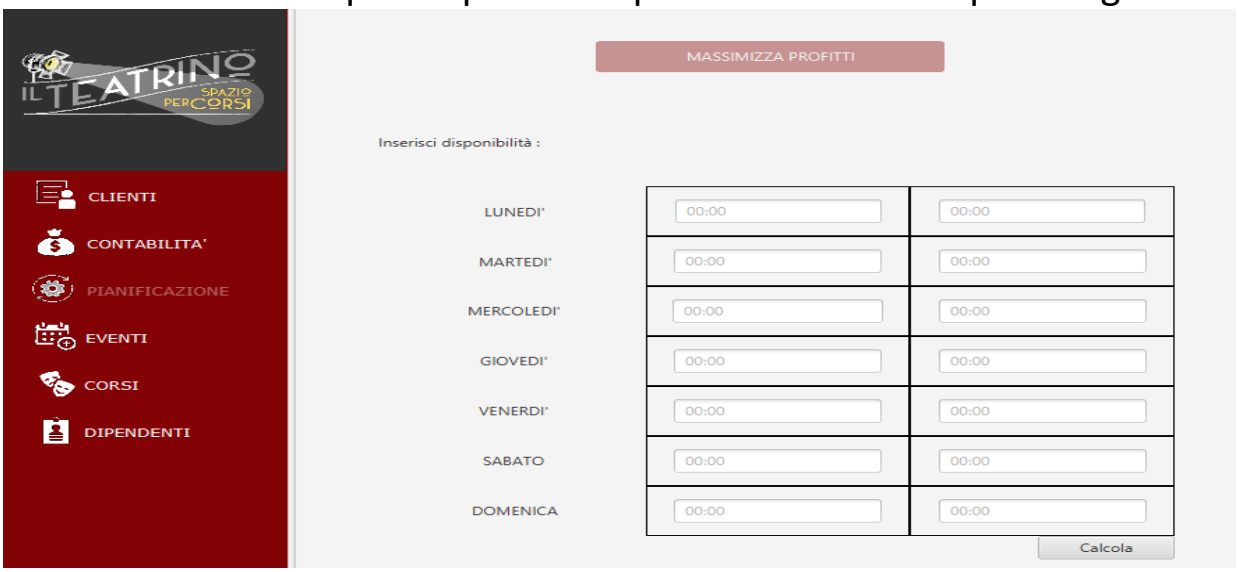


6. FUNZIONAMENTO DELL’ APPLICAZIONE

All’apertura dell’applicazione viene mostrato un menù.



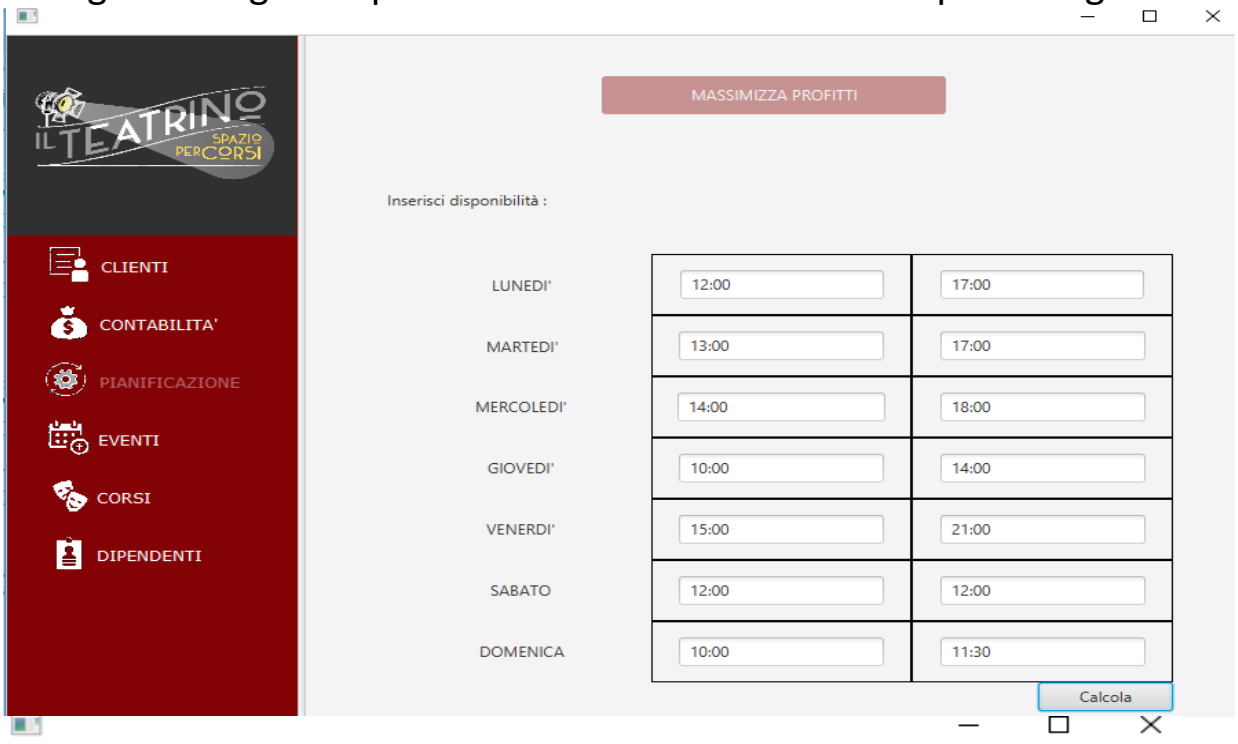
Selezionando uno dei diversi bottoni si aprono schermate relative al testo del bottone. Viste le numerose possibilità mi concentrerò solamente sulla parte dell’applicazione che maggiormente concerne con la tesi. Perciò si ipotizza che l’utente clicchi il tasto “Pianificazione”. A questo punto si aprirà la schermata qui di seguito.



Come già visto nel punto 4, qui l’utente inserisce le disponibilità e cliccando su “Calcola” fa partire il ricorsivo che restituirà la sequenza ottimale.

7. RISULTATI SPERIMENTALI OTTENUTI

Di seguito vengono riportati i risultati ottenuti con l'input di seguito.



CORSO	GIORNO	ORARIO
Orlando	Lunedì	12:00 - 13:30
Adulti II	Lunedì	13:30 - 15:30
Medie	Lunedì	15:30 - 17:00
Superiori	Martedì	13:00 - 14:30
Giuliette&Romei	Martedì	14:30 - 16:00
Adulti I	Mercoledì	14:00 - 16:00
Universitari	Mercoledì	16:00 - 18:00
Puck	Giovedì	10:00 - 11:30
Piccolissimi	Martedì	16:00 - 17:00
Favolosi	Venerdì	15:00 - 16:00
Profitto settimanale previsto : 770 €		

Link al video dimostrativo: https://youtu.be/pLvPllKYWZc

8. VALUTAZIONI E CONCLUSIONI

I principali problemi di implementazione erano principalmente legati al poter rimuovere e reinserire un corso nella sequenza, superato questo problema risultava molto complicato andare a creare una sequenza senza sovrapposizione dei corsi. La creazione della classe Giorno ha risolto questo problema. Una volta superati questi limiti l'applicativo ha mostrato grande fluidità.

In particolare i numerosi vincoli (giorni liberi dei dipendenti, impossibilità di reinserire un corso in un girono in cui è già stato , impossibilità di corsi prima delle 7 AM o dopo le 11 PM) permettono grandi prestazione. L'utente ottiene infatti la sequenza in tempi brevissimi (millesimi di secondo) poiché le sequenze che vengono trovate non sono mai superiori a 6/7. In questa ottimizzazione è stato fondamentale il vincolo per cui un corso non può essere reinserito in un giorno (anche perché l'output del metodo getProfitto() è maggiormente sensibile rispetto a variazioni di giorni piuttosto che di ore).

L'applicazione risulta quindi molto rilevante per la titolare de "Il Teatrino" che potrà contare su uno strumento utile sia per far coincidere le varie disponibilità, sia per ottenere i massimi profitti possibili.

Inoltre l'intera applicazione permette non solo di gestire e visualizzare dati, ma anche di modificarli ,aggiungerli o eliminarli. Da sottolineare anche la parte visuale che permetterà all'utente di cogliere rapidamente i guadagni e le perdite relative a corsi e/o dipendenti.

Il software risulterà quindi molto utile, sia dal punto di vista della gestione dati (andando ad alleggerire notevolmente il lavoro dell'utente) sia dal punto di vista della pianificazione e della ricerca di una strategia che permetta di guadagnare sempre di più.