SmartRistò

Preparato da: Oleksandr Melnychuk , Luigi Giddio, Patrick Noumsi.

RIEPILOGO ESECUTIVO

Scopo

Il gruppo si pone come obiettivo quello di realizzare un software per la gestione dei ristoranti.

Obiettivi

Il software sarà in grado di gestire: gli ordini dei tavoli direttamente da tavoli siccome la nostra idea è di vere un dispositivo su ogni tavolo in modo che il cliente faccia l'origine in modo autonomo, le disposizioni dei tavoli nella sala eventuale modifica e eliminazione.

Il software esporrà le seguenti funzionalità:

- gestione ordini dei tavoli
- aggiunta/modifica/eliminazione dei tavoli
- possibilità di emettere lo scontrino
- gestione del menu
- gestione della sala (tavoli occupati, liberi, prenotati)
- prenotazione dei tavoli

Feature opzionali:

- possibilità di vari metodi di pagamento
- gestione del personale

Struttura del progetto

Durante lo sviluppo si è scelto di utilizzare il pattern MVP per ottenere una chiara suddivisione dei compiti. Model-View-Presenter (MVP) è una derivazione del modello-View-Controller (MVC) modello architettonico, ed è utilizzato soprattutto per la costruzione di interfacce utente.

MVP è un pattern architetturale interfaccia utente progettata per facilitare il test di unità automatizzati e migliorare la separazione degli interessi in logica di presentazione:

Il modello è un'interfaccia che definisce i dati da visualizzare o altrimenti agito su nell'interfaccia utente.

Il presentatore agisce sul model e la view. Si recupera i dati dal repository (il modello), e formatta per la visualizzazione nella view.

La view è un interfaccia passiva che visualizza i dati (il modello) e i comandi percorsi utente (eventi) per il Presenter che agisce su quei dati.

Il progetto è costituito dai vari package che verrano illustrati in dettaglio qui in seguito.
Un esempio :

```
public class Model
{
    public String getWhatIWantToSay()
        {
            return "Hello World";
        }
}
```

```
public class Presenter implements ActionListener
{
  private final View view;
  private final Model model;
  public Presenter(Model model, View view)
        {
            this.model = model;
            this.view = view;
        view.addButtonListener(this);
        }
  public void actionPerformed(ActionEvent e) {
```

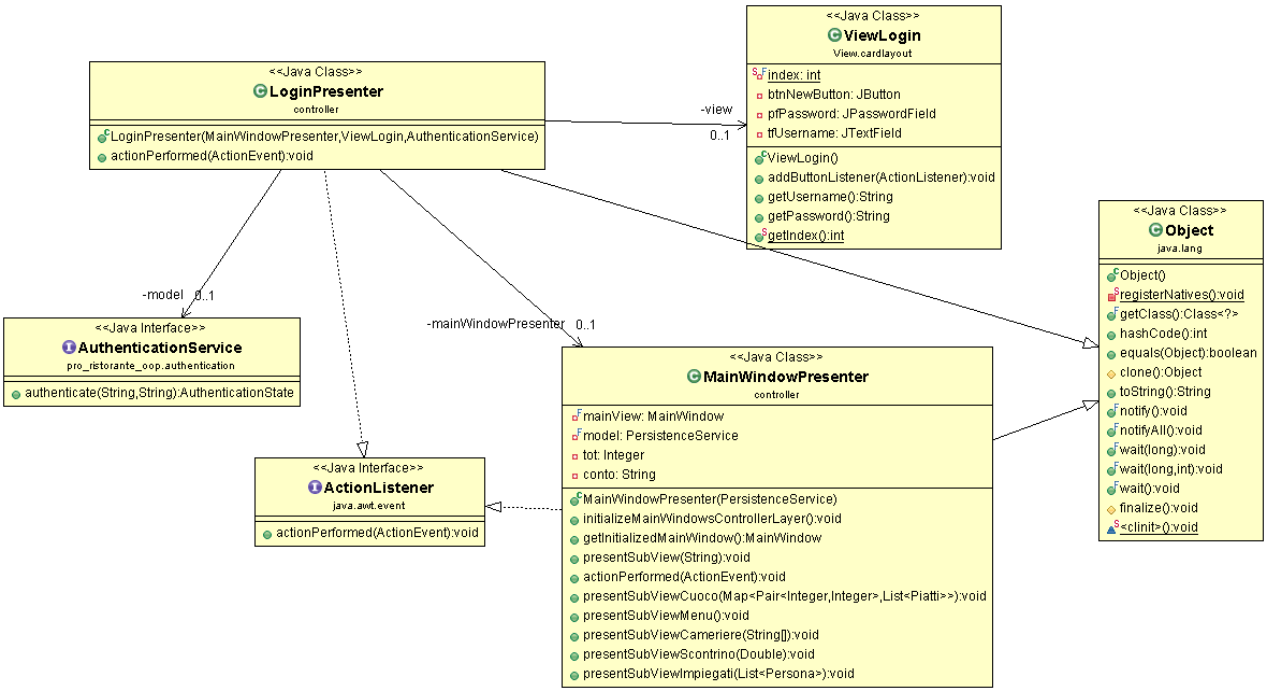
```
    view.setText(model.getWhatIWantToSay());
    }
}

public class View
{
    private JButton button = new JButton();
    private JLabel label = new JLabel();
    public void addButtonListener(ActionListener listener)
     {
        button.addActionListener(listener);
    }
    public void setText(String text)
     {
        label.setText(text);
    }
}
```

- ## PACKAGE CONTROLLER

Questo package contiene le classi LoginPresenter,MainWindowPresenter,MenuPresenter,PrenotationPresenter,CuocoPresenter,CamerierePresenter,ScontrinoPresenter.

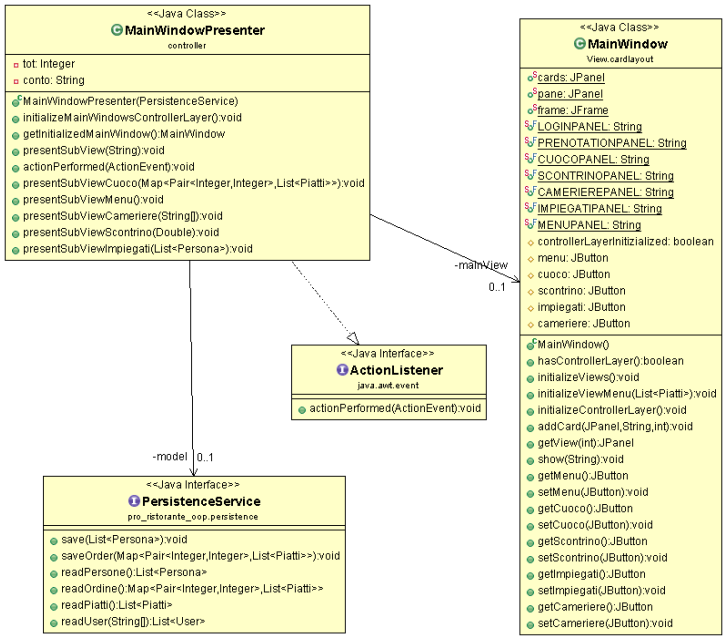
LoginPresenter

Questa classe implementa ActionListener e gestisce la prima view che viene lanciata cioè quella del login, collegandosi con view e il model tramite le classi AuthenticationService (questo controlla lo stato del login se è andato a buon fine) e ViewLogin alla pressione del bottone del Login viene inizializzata la MainView, insieme alla view Prenotazione.



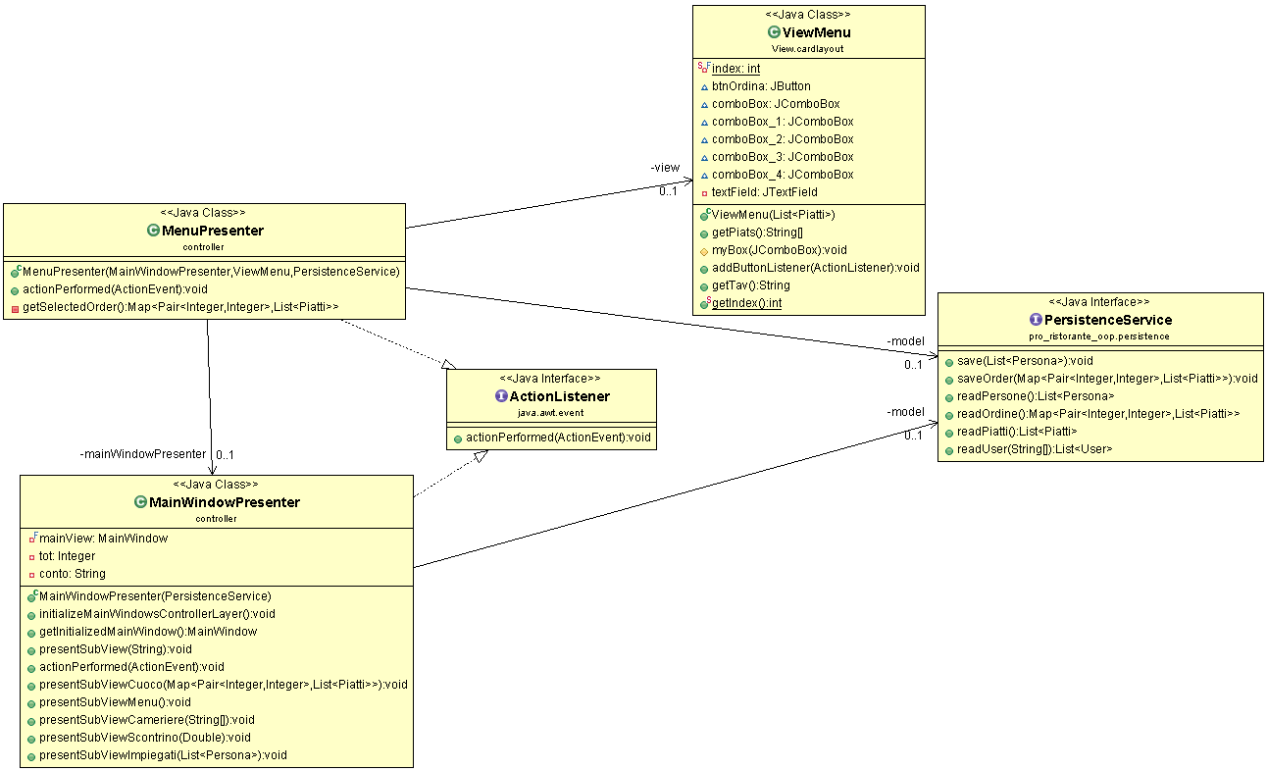
MainWindowPresenter

Questa classe gestisce la ceretta visualizzazione delle view attraverso vari bottoni, cioè qui si inizializza la MainView che avrà come il contenuto altre View che dipenderanno dalla scelta del bottone e di default contiene la view delle Prenotazioni. Qui si caricano i piatti necessari per la creazione del Menù dal file Piatti.txt richiamando i metodi presenti nel model che sono presenti nel package pro_ristorante_oop.persistance.

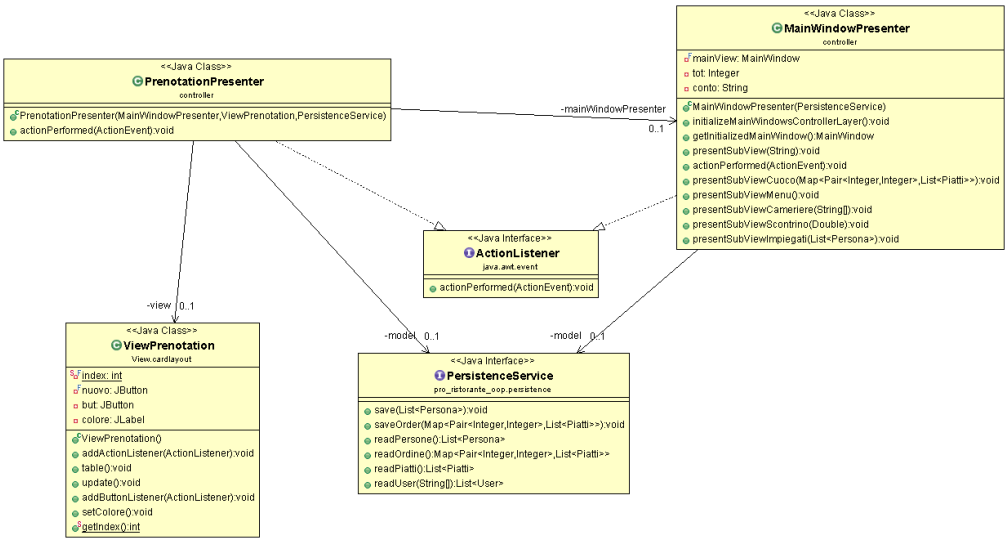


MenuPresenter

Questa classe gestisce la selezione dei piatti dal menù e il seguente salvataggio del ordine eseguito. Il metodo getSelectedOrder permette di salvare l'ordine selezionato le file ordini.txt alla pressione ordine e attraverso actionPerformed() di passare alla viewCuoco con gli ordini che vengo passati direttamente alla view.



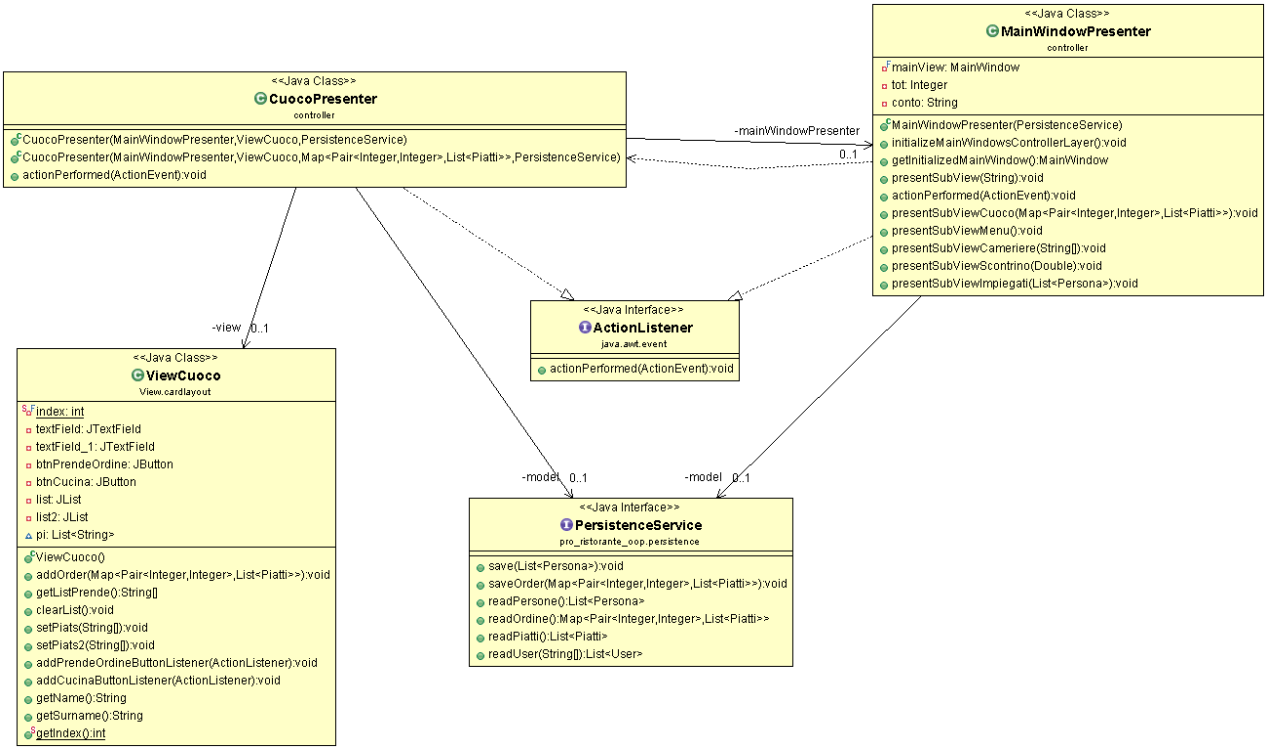
PrenotationPresenter



Questa classe gestisce le prenotazioni dei tavoli cioè la creazione di nuovo tavolo e la sua prenotazione attraverso actionPerformed(). Alla pressione new Client si crea un nuovo tavolo alla pressione prenota si prenota un tavolo.

CuocoPresenter

Questa classe gestisce ordini ordinati cioè ordini già effettuati dai clienti, quali vengono caricati nella lista, e poi passati alla ViewCameriere, attraverso i pulsanti “prende ordine” e “cucina”.

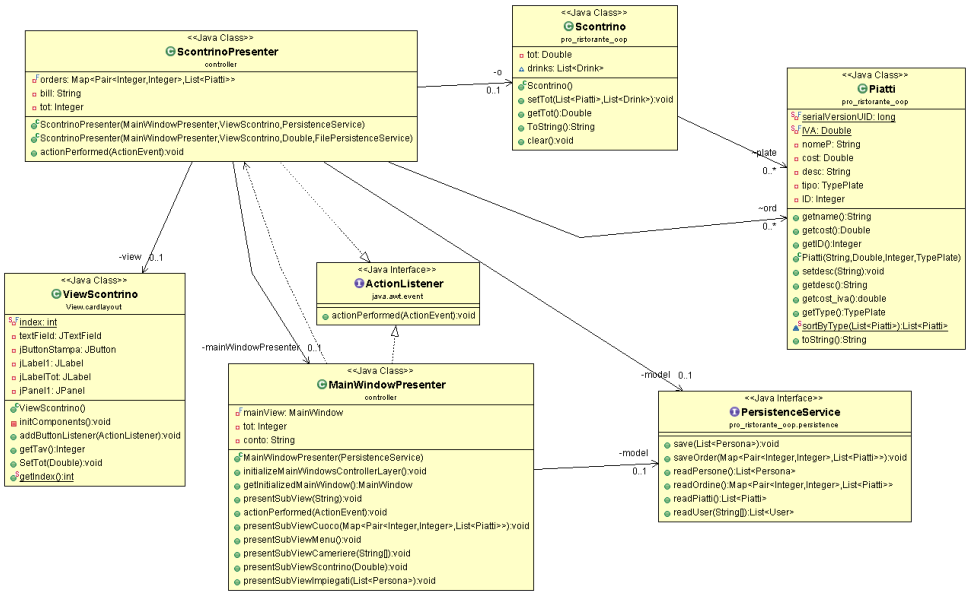


CamerierePresenter

Questa classe gestisce gli ordini eseguiti dal Cuoco che sono pronti per essere portati in tavola, che vengono visualizzati nella lista del Cameriere Presente nella view.

Scontrino Presenter

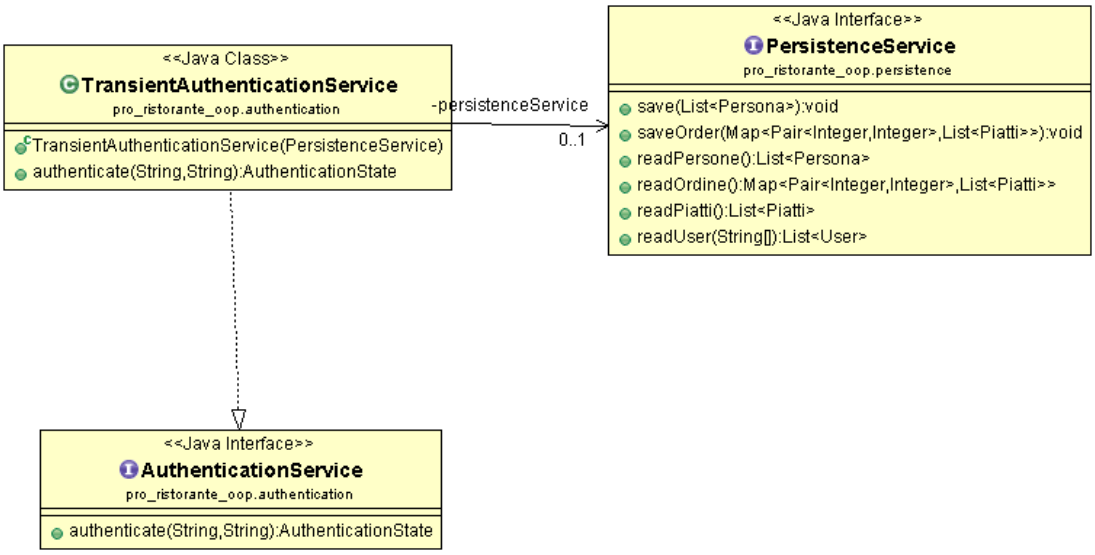
Fornisce la possibilità di stampare lo scontrino attraverso il caricamento del ordine dal file e con il metodo setTot del model che permette di calcolare il totale delle pietanze. Alla pressione del pulsante stampa viene



visualizzato il totale.

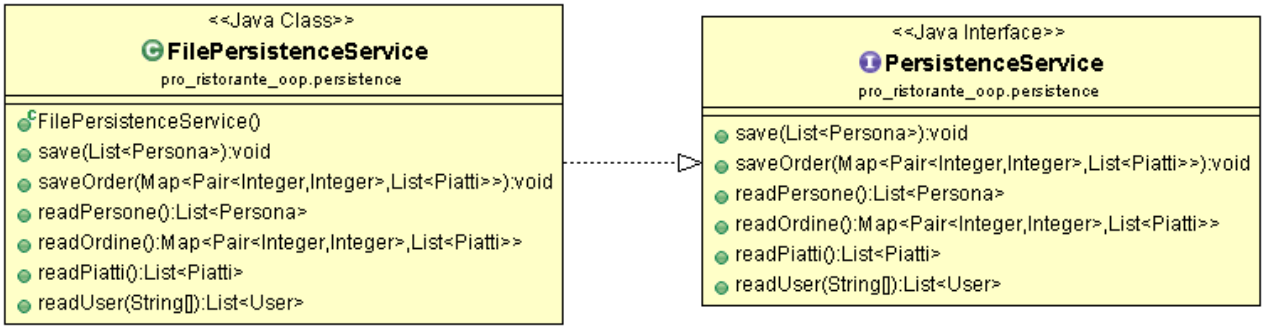
Pro_ristorante_opp.authentication

Fornisce la possibiltà di controllo sul login. Per logica concettuale l'ho messa nel model ma è stata ideata dal controller.



Pro_ristorante_oop.persistance

Fornisce la possibilità di gestione dei file di salvataggio e caricamento dei file che servono per la gestione dei ordini, piatti, menu ,persone,User. Per logica concettuale l'ho messa nel model ma è stata ideata dal controller.



• PACKAGE VIEW

ViewLogin

public ViewLogin() questa classe crea la view di entrata e permette di mettere login e password a traverso i due metodi public String getPassword() e public String getUsername() questi metodi permettono di catturare gli input nei campi di username e password che successivamente serviranno al controller per il controllo dei dati del login. public static int getIndex() restituisce l'indice della view.

MainWindow

Questa classe definisce i metodi per la visualizzazione delle view dentro la scremata principale e contiene di default la view di prenotazione.

public void initializeViews() serve per inizializzare tutte le view che verrano visualizzate e viene usato un altro metodo addCard() che serve per aggiungere delle view alla MainWindow il quale in parametro a un Jpanel , una Stringa E un componentIndex.

public void initializeViewMenu(List<Piatti> piatti) serve per creare la prima view come default che viene visualizzata per prima con i piatti gi‡ caricati dal controller.GetView() restituisce un jpanel e poi ci sono i vari metodi Get() e Set() per ogni Jbutton che rappresenta la lista delle view che si possono visualizzare all' interno della MainWindows.

public void show(String card) permette di visualizzare il CardLayout.

ViewMenu

Questa classe contiene il Jpanel del menu che serve per la visualizzazione del menu del ristorante con il metodo public ViewMenu(List<Piatti> piatti) il quale inizializza questo jpanel e in input prende una lista di piatti passata dal controller la quale verrà visualizzata nelle varie combo box a seconda del piatto.

public void addButtonListener(ActionListener listener) aggiunta del listener.

public String getTav() restituisce il numero del tavolo inserito.

public static int getIndex() restituisce l'indice della view.

public String[] getPiats() restituisce i piatti che sono stati selezionati nelle combo box.

protected void myBox(JComboBox comboBox) controlla se non Ë stato selezionato nessun piatto.

ViewPrenotazione

Questa classe contiene il Jpanel delle diverse prenotazione che serve per la visualizzazione dei vari tavoli e clienti del ristorante con il metodo.

public void table() assegna un tavolo ad ogni nuovo cliente.

public void update() aggiorna la MainView per fare comparire i nuovi tavoli creati.

public void setColore() cambia seconda dello stato degli ordini su ogni tavolo Verde se non hanno ancora ordinato i clienti e Rosso nel caso contrario.

public static int getIndex() restituisce l'indice della view.

ViewCuoco

questa classe serve a visualizzare la lista dei piatti che deve cucinare il cuoco e da mandare al cameriere.

public ViewCuoco() inizializza il Jpanel.

public void addOrder(Map<Pair<Integer,Integer>,List<Piatti>> ordine) legge i dati dalla mapa ordine e lo mette nella lista del cuoco.

public void setPiats(String[] val) e public void setPiats2(String[] val) aggiungono dei valori alle liste del cuoco.

public void addPrendeOrdineButtonListener(ActionListener listener) e public void addCucinaButtonListener(ActionListener listener) sono i due listener dei pulsanti.

public static int getIndex() restituisce l'indice della view.

ViewCameriere

questa classe serve per la visualizzazione dei piatti che devo essere portati al tavolo.

public ViewCameriere() inizializza il Jpanel.

public static int getIndex() restituisce l'indice della view.

public void addButtonListener(ActionListener listener) cambia il colore dello stato del cameriere a seconda che sia libero o occupato.

ViewScontrino

questa classe imposta il Jpanel per la visualizzazione dello scontrino.

public Integer getTav() serve per calcolare il totale del ordine su un tavolo.

public void SetTot(Double tot) restituisce il totale da pagare.

public static int getIndex() restituisce l'indice della view.

CONCLUSIONI

Lo sviluppo non è stato lineare e non siamo molto soddisfatti per quanto riguarda l'implementazione di tutte le funzionalità richieste dall'applicazione. Per motivi di tempo non si è riuscito ad implementare la funzionalità in modo coretto ed efficace.Per quanto riguarda la progettazione penso che abbiamo effettuato una buona progettazione parlando di struttura del codice ma che purtroppo non siamo riusciti a struttura al meglio. Sono soddisfatto dell'esperienza del lavoro di gruppo.

MODEL

Package pro_ristorante_oop

Questo package contiene le classi Cameriere,Cuoco,Proprietario,Ordine,OrdineD,Piatti,Drink, Tavoli,Scontrino,SurplusSeatsExceptions,Pair,Persona.
Le interfacce IScontrino,ITavoli,IOrdine,IOrdineD,IPiatti.
L' enumerazione TypePlate.

Persona

Classe astratta implementata in Cameriere,Cuoco,Proprietario perchè contiene metodi comuni a tutte e tre.

Cameriere

Questa classe eredita la classe astratta Persona,contiene metodi per la gestione dell'utente Camerieri del ristorante,gestendo i piatti e gli ordini dei tavoli, inoltre contiene le credenziali per riconoscere l'utente attraverso gli attributi nome,cognome,password.

Cuoco

Questa classe eredita la classe astratta persona,è utilizzata per gestire i piatti ordinati dai tavoli e quelli cucinati pronti per essere portati via.
Come la classe cameriere permettere di riconoscere l'utente tramite determinati attributi.

Proprietario

Questa classe eredita la classe astratta persona ed è utilizzata solo per poter riconoscere il proprietario da gli altri dipendenti.

Tavoli

Questa classe implementa l'interfaccia ITavoli, gestisce i clienti al tavolo e le loro ordinazione. Controllo il numero di clienti al tavolo e vede di inviare lo ordinazioni al cuoco o al cameriere.

Scontrino

Questa classe implementa l'interfaccia IScontrino e gestisce il conto del tavolo facendo vedere al cliente la spesa di tutte le ordinazioni e il prezzo dei singoli piatti.

Ordine

Implementa l'interfaccia IOrdine,questa classe gestisce tutte le ordinazione dei tavoli raggruppandole in una Map.
Una volte che il tavolo effettua un ordine viene messo in coda nella Map in attesa che il cuoco prenda un ordinazione .

OrdineD

Implementa l'interfaccia IOrdineD,questa classe gestisce gli ordini pronti per essere consegnati ai tavoli.
Questi ordini sono messi in attesa fino a quando un cameriere non è libero e pronto a prendere i piatti.

Drink

Implementa l'interfaccia IPiatti,questa classe gestisce le bevande che il cameriere deve portare al tavolo.
Come i piatti sono messi nello scontrino con il relativo prezzo.

Piatti

Implementa l'interfaccia IPatti,questa classe gestisce le pietanze che i clienti ordinano.
Sono dotati di un tipo (typeplate) che permette al sistema di potergli ordinare in base all'importanza.

TypePlate

Un'enumerazione che permette di distinguere i piatti in Antipasti,Primi,Secondi.

Pair

Una classe presa da un altro package e permette di creare un'istanza con due diversi tipi.
Nel software e utilizzata negli ordini per poter gestire più ordine dallo stesso tavolo.

SurplusSeatsException

Implementa l'interfaccia Exception e solleva un'eccezione quando si vuole inserire un numero di clienti maggiore rispetto al numero di posti disponibili.