SAPIENZA
UNIVERSITÀ DI ROMA

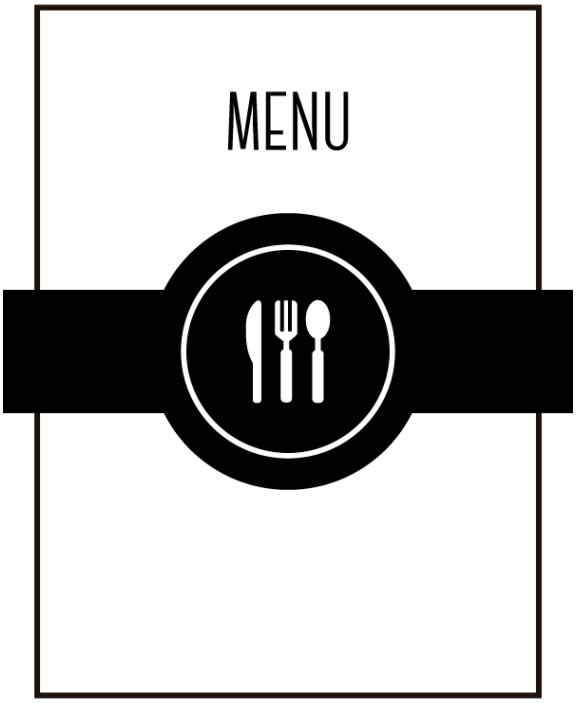
DIPARTIMENTO INGEGNERIA DELL'INFORMAZIONE, INFORMATICA E STATISTICA

UNIVERSITÀ DI ROMA LA SAPIENZA

CORSO DI INFORMATICA

METODOLOGIE DI PROGRAMMAZIONE

GESTORE RISTORANTE



Authors: Germani Patrizio, Daniele Fasano, Alessia Ciarla

Teacher: Walter Quattrociocchi

ACADEMIC YEAR 2020-2021

Indice

1

Introduzione

Il nostro progetto consiste nel realizzare l'intera gestione di un piccolo ristorante, in modo da poter compiere tutte le operazioni principali necessarie in modo corretto, a partire dall'organizzazione dell'intero menù all'emissione degli scontrini, tutto in maniera digitale.
Si partirà da un menù principale, dal quale si potranno selezionare quattro ruoli, per la normale gestione del locale.
Il primo è "CHEF", grazie al quale si potrà ideare l'intero menù e aggiungere, modificare e rimuovere i piatti al suo interno.
Il secondo, "CAMERIERE", visualizzerà la lista dei tavoli disponibili nel ristorante e prenderà le rispettive ordinazioni, selezionando poi i piatti desiderati dai clienti, con le relative quantità.
Il Terzo è "CUOCO", il quale visualizzerà solo i tavoli per cui gli ordini sono stati già inseriti precedentemente dal cameriere e, dopo aver scelto il tavolo interessato e aver visualizzato la relativa comanda, l'ordine verrà evaso e sarà pronto per essere finalizzato con il ruolo successivo, "CASSA".
Con quest'ultimo ruolo, allo stesso modo, si potrà scegliere solo tra i tavoli che sono stati già evasi nella sezione cuoco; quindi, una volta scelto il tavolo desiderato, si rende poi visibile lo scontrino con il prezzo totale, al fine di finalizzare il pagamento.
Si è cercato il più possibile di attuare una partizione del lavoro sempre equa, in modo da non lasciare mai un membro del gruppo senza un compito assegnato; la suddivisione dei compiti è stata fatta, a grandi linee, secondo le varie classi del programma.
Il codice, che è stato scritto con l'editor Eclipse, è costituito anche da un'interfaccia grafica (GUI), creata tramite la libreria Swing.
Al fine di ottenere una collaborazione diretta tra i membri del gruppo abbiamo utilizzato Git e GitHub per quanto riguarda il codice, mentre la relazione è stata scritta in LaTeX con l'utilizzo di Overleaf.

2

Descrizione delle Classi

Innanzitutto, bisogna dire che tutto il progetto è racchiuso nel package ”gestoreRistorante”, che contiene quindi le classi servite a realizzare la schermata di partenza e quattro package, rispettivamente uno per ogni ruolo implementato, che a loro volta contengono le classi impiegate nella realizzazione degli stessi.

2.1 package gestoreRistorante

Il package GestoreRistorante è, gerarchicamente, il più esterno e racchiude tutti i package e le classi relative al progetto.

2.1.1 classe MenuPrincipale

La classe MenuPrincipale è posta all'interno del package *gestoreRistorante* e implementa la parte grafica del menù iniziale, da cui è possibile scegliere un ruolo tra: chef, cameriere, cuoco e cassa tramite quattro bottoni.
Gli *attributi* della seguente classe sono:

- **COLORE_SFONDO**: variabile statica utilizzata per lo sfondo dei frame e dei pannelli dell'applicazione;
- **COLORE_BOTTONI**: variabile statica utilizzata per lo sfondo dei bottoni dell'applicazione;
- **start**: viene inizializzato un oggetto JFrame con questo nome;
- **contenuto**: viene creato un oggetto di tipo Container che rappresenta il contenuto del frame start;
- **tavoli**: viene creato un oggetto di tipo *ListaTavoli*, che servirà nel caso in cui venisse cliccato il bottone in basso, nella schermata di partenza, *”CHIUDI SESSIONE”*, per reimpostare i tavoli al loro stato iniziale.

I *metodi* della seguente classe sono:

- **MenuPrincipale()**: metodo costruttore che richiama il metodo *visualizza()*;
- **visualizza()**: metodo che imposta la grandezza del frame e crea la grafica relativa alla prima schermata dell'applicativo, costituita da cinque bottoni(quattro per le categorie e uno per la chiusura della sessione in corso).

2.1.2 CLASSE TESTERRISTORANTE

Anch'essa è racchiusa nel package *gestoreRistorante* ed è la classe grazie alla quale è possibile eseguire il programma e visualizzare graficamente l'applicazione realizzata.

- l'unico oggetto creato in questa classe è di tipo *MenuPrincipale*;
- l'unico metodo presente nella classe è il metodo principale **main**, all'interno del quale viene creato l'oggetto di tipo *MenuPrincipale*, grazie al quale è possibile eseguire il programma.

2.1.3 INTERFACCIA LISTA

Interfaccia che viene utilizzata per implementare il polimorfismo nelle classi che utilizzano il metodo *write()*.

- l'unico metodo creato è il metodo astratto **write()**, che serve per sovrascrivere un determinato file.

2.2 PACKAGE CHEF

Il package *chef* si trova all'interno del package *gestoreRistorante*, e contiene tutte le classi inerenti al funzionamento del primo bottone, correlato al ruolo di chef. In questo package sono presenti sia la parte grafica di front-end, che quella di back-end.

2.2.1 CLASSE LISTAPIATTI

Classe di back-end che, in sostanza, contiene tutti i piatti all'interno del menù (tramite un ArrayList); inoltre, vengono creati anche i metodi che saranno usati nella classe MenuChef(che è invece di front-end), per gestire la parte grafica.
Gli *attributi* della seguente classe sono:

- **listapiatti**: ArrayList di tipo Piatto, che contiene tutti i piatti del menù che crea lo chef;

- **file**: variabile di tipo File, in cui viene salvato il file *"menu.txt"*, preso in input, che viene popolato con i piatti che vengono aggiunti graficamente al menù (file persistente).

I *metodi* della seguente classe sono:

- **ListaPiatti()**: metodo costruttore che richiama il metodo *read()*;
- **add(Piatto datiPiatto)**: metodo che aggiunge un nuovo piatto, di tipo *Piatto*, all'ArrayList, evitando di inserire duplicati;
- **sort()**: metodo che ordina l'ArrayList tramite una *Collections.sort(listapiatti, new OrdinaPiatti())*, che prende come parametri l'ArrayList *listapiatti* con tutti i piatti del menù, e un nuovo oggetto di tipo *OrdinaPiatti()*(vedi 2.2.3);
- **clear()**: metodo che rimuove tutto il contenuto dell'ArrayList;
- **remove(Piatto datiPiatto)**: metodo che rimuove il piatto, passato come parametro, dall'ArrayList;
- **getPiatto(int indice)**: metodo grazie al quale è possibile ricavare un piatto, di tipo Piatto, nell'Arraylist listapiatti;
- **size()**: metodo grazie al quale si ricava la lunghezza dell'Arraylist listapiatti;
- **modify(Piatto dasostituire, Piatto sostituto)**: metodo che permette di sostituire o modificare i dati di un certo piatto(nome, prezzo o categoria), dove *dasostituire* è il piatto da modificare, mentre *sostituto* è il piatto che va inserito al posto del precedente;
- **read()**: metodo che legge dal file *"menu.txt"* in cui è contenuto il menù e lo copia all'interno dell'ArrayList *listapiatti*(effettua quindi il passaggio dal file persistente alla grafica);
- **write()**:metodo che legge il contenuto dell'ArrayList *listapiatti* e lo copia all'interno del file *"menu.txt"*(effettua quindi il passaggio dalla grafica al file persistente).

2.2.2 CLASSE MENUCHEF

Classe di front-end che implementa la grafica del menù che visualizzerà lo chef, che sarà modificabile grazie ad alcuni bottoni.
Gli *attributi* della seguente classe sono:

- **categorie**: Array statico di stringhe che contiene le cinque categorie fisse del menù(ANTIPASTI, PRIMI, SECONDI, CONTORNI,DOLCI);
- **listap**: oggetto di tipo *ListaPiatti*, il quale contiene tutti i piatti del menù;
- **editable_menu**: oggetto di tipo JFrame, che consiste nel frame del menù dello chef, che sarà editabile;
- **contenuto**: oggetto di tipo Cointainer, che è il relativo ContentPane di *editable_menu*;
- **pannello_variabile**: oggetto di tipo JSplitPane, che consiste nel pannello centrale che conterrà graficamente il menù, che potrà variare eliminando, aggiungendo o modificando i piatti.

I *metodi* della seguente classe sono:

- **MenuChef()**: metodo costruttore che richiama il metodo *visualizza()*;
- **visualizza()**: metodo che imposta la grandezza del frame, la parte grafica alta della schermata relativa al menù dello chef e, inoltre, richiama il metodo *popolaPannello()*;
- **popolaPannello()**: metodo che, come da nome, è in grado di popolare il pannello centrale(attributo denominato come *pannello_variabile*) con vari oggetti;
- **aggiungiPiatto(String nome_piatto, double prezzo_piatto, int category)**: metodo che permette di aggiungere un piatto sia graficamente che funzionalmente(sia back-end che front-end); per i parametri possiamo notare *nome_piatto*, *prezzo_piatto* e *category*, che rispettivamente identificano il nome, il prezzo e la categoria del piatto che si vuole aggiungere;
- **rimuoviPiatto(String nome_piatto, double prezzo_piatto, int category)**: metodo che permette di rimuovere un piatto sia graficamente che funzionalmente(sia back-end che front-end); per i parametri possiamo notare *nome_piatto*, *prezzo_piatto* e *category*, che rispettivamente identificano il nome, il prezzo e la categoria del piatto che si vuole eliminare dal menù;
- **modificaPiatto(Piatto dasostituire, Piatto sostituto)**: metodo che consente di modificare un piatto, confrontando il vecchio piatto da sostituire(primo parametro), con il nuovo piatto con i dati modificati(secondo parametro);

- **utilizzaPolimorfismo(Lista po)**: metodo che utilizza il polimorfismo, per scrivere sui vari file dell'applicazione, quando richiesto; il parametro passato è di tipo *Lista*, che è un'interfaccia.

2.2.3 classe OrdinaPiatti

Classe di back-end che implementa l'interfaccia *Comparator*, e grazie al quale è possibile confrontare due piatti, con le rispettive categorie(tramite un intero che svolge il ruolo di identificativo).

- Questa classe ha un unico metodo, che è il metodo **compare(Piatto o1, Piatto o2)**, che viene utilizzato(tramite *Override* del metodo) per confrontare due oggetti di tipo *Piatto*. Poi, questo metodo, viene utilizzato nella classe *ListaPiatti*, e più precisamente nel metodo *sort()*(vedi 2.2.1).

2.2.4 classe Piatto

Classe di back-end grazie alla quale viene identificato un piatto all'interno del menù.
Gli *attributi* della seguente classe sono:

- **name**: stringa che si riferisce al nome del piatto;
- **price**: double che si riferisce al prezzo del piatto;
- **category**: intero che si riferisce alla categoria a cui appartiene il piatto(si intende 0 per gli antipasti, 1 per i primi, 2 per i secondi, 3 per i contorni e 4 per i dolci);

I *metodi* della seguente classe sono:

- **Piatto(String nome, double prezzo,int numcat)**: metodo costruttore che associa i tre parametri passati in input agli attributi della classe Piatto;
- **getPrice()**: metodo che ritorna il prezzo associato ad un piatto;
- **setPrice()**: metodo che imposta il prezzo da associare ad un piatto;
- **getName()**: metodo che ritorna il nome del piatto;
- **setName()**: metodo che imposta il nome del piatto;
- **getNumcategory()**: metodo che ritorna l'intero identificativo della categoria di un certo piatto;

- **setNumcategory()**: metodo che imposta l'intero identificativo della categoria di un certo piatto;
- **equals(Object obj)**: questo metodo è caratteristico di tutte le classi, ma bisogna implementarlo(tramite *Override* del metodo) se si necessita usarlo, essendo Piatto una nuova classe creata; è utile al fine di confrontare due oggetti di tipo Piatto.

2.3 PACKAGE CAMERIERE

Il package *cameriere*, che si trova all'interno del package *gestoreRistorante*, contiene tutte le classi inerenti al funzionamento del secondo bottone correlato al ruolo di cameriere. In questo package ci sono sia la parte grafica di front-end che quella di back-end.

2.3.1 CLASSE ELENCOTAVOLICAMERIERE

Classe di front-end che implementa la gestione dei tavoli da parte del cameriere, cioè permette di visualizzare la lista di tutti i tavoli del ristorante e il loro stato attuale.
Gli *attributi* della seguente classe sono:

- **nomitavoli**: Array statico di stringhe che contiene i cinque tavoli fissi del ristorante(TAVOLO1, TAVOLO2, TAVOLO3, TAVOLO4, TAVOLO5);
- **listat**: oggetto di tipo *ListaTavoli*, grazie al quale è possibile mantenere tutti i tavoli in un ArrayList, su cui si possono applicare poi i metodi;
- **table_view**: viene creato graficamente un nuovo JFrame, con il rispettivo ContentPane;
- **pannello_centrale**: pannello centrale che conterrà i cinque bottoni relativi ai tavoli e il loro status;

I *metodi* della seguente classe sono:

- **ElencoTavoliCameriere()**: metodo costruttore che richiama il metodo *visualizza()*;
- **visualizza()**: crea la parte alta della finestra e imposta le specifiche grafiche del pannello principale;
- **popolaPannello()**: è in grado di popolare il pannello centrale con vari oggetti.

2.3.2 classe ListaTavoli

Classe di back-end che implementa l'ArrayList dei tavoli e, inoltre, gestisce vari metodi che saranno usati nelle altre classi di front-end contenute nel package *cameriere*.
Gli *attributi* della seguente classe sono:

- **listatavoli**: ArrayList che identifica la lista dei tavoli;
- **file**: identifica il file su cui andare a scrivere l'elenco dei tavoli con il relativo stato.

I *metodi* della seguente classe sono:

- **ListaTavoli()**: metodo costruttore che richiama il metodo *read()*;
- **getTavolo(int indice)**: metodo grazie al quale è possibile ricavare un oggetto di tipo *Tavolo* dall'Arraylist *listatavoli*;
- **size()**: metodo grazie al quale si ricava la lunghezza dell'Arraylist *listatavoli*;
- **read()**: metodo grazie al quale è possibile leggere il contenuto del file *"lista_tavoli.txt"*, in cui è contenuta la lista dei tavoli, e copiarlo all'interno dell'ArrayList *listatavoli*;
- **write()**: metodo che legge il contenuto dell'ArrayList *listatavoli* e lo scrive all'interno del file *"lista_tavoli.txt"*, nel caso in cui lo stato di un tavolo cambi.

2.3.3 classe Ordinazione

Classe di back-end che contiene l'ArrayList di tutti i piatti ordinati, e gestisce vari metodi che saranno usati nella classi *TavoloSingolo* e *RiepilogoCameriere*.
Gli *attributi* della seguente classe sono:

- **listapiatti**: ArrayList contenente la lista dei piatti ordinati;
- **file**: attributo di appoggio dove salvare momentaneamente l'ordinazione.

I *metodi* della seguente classe sono:

- **Ordinazione()**: metodo costruttore che richiama il metodo *read()*;
- **add(Piatto datiPiatto)**: metodo che aggiunge un piatto all'interno dell'ArrayList dei piatti ordinati;
- **sort()**: metodo grazie al quale è possibile ordinare l'ArrayList dei piatti ordinati tramite una *Collection.sort*;

- **clear()**: metodo che rimuove tutto il contenuto dell'ArrayList *listapiatti*;
- **getPiatto(int indice)**: metodo grazie al quale è possibile ricavare un piatto di tipo Piatto nell'Arraylist listapiatti;
- **size()**: metodo che ricava la lunghezza dell'Arraylist listapiatti;
- **read()**: metodo grazie al quale è possibile leggere dal file *"appoggio.txt"* in cui è contenuta l'ordinazione e copiarla all'interno dell'ArrayList *listapiatti*;
- **write()**: metodo grazie al quale è possibile leggere il contenuto dell'ArrayList *listapiatti* e copiarlo all'interno del file *"appoggio.txt"*.

2.3.4 classe RiepilogoCameriere

Classe di front-end che implementa la grafica dell'ordinazione finale di un tavolo, che visualizzerà il cameriere.
Gli *attributi* della seguente classe sono:

- **tavoli**: oggetto di tipo *ListaTavoli*, grazie al quale è possibile mantenere tutti i tavoli in un ArrayList;
- **riepilogo**: viene creato graficamente un nuovo JFrame, con il rispettivo ContentPane;
- **pannello_centrale**: viene creato come attributo anche il pannello che conterrà l'ordinazione finalizzata del tavolo scelto;
- **listap**: oggetto di tipo *Ordinazione*, grazie al quale riesco ad avere tutte le quantità di ogni piatto ordinato;
- **inFile**: file in cui si copia tutta l'ordinazione in modo tale da poterla utilizzare successivamente per lo scontrino;
- **numerotavolo**: attributo di tipo intero che rappresenta il numero del tavolo su cui sto prendendo l'ordinazione, per poter aggiornare il suo stato.

I *metodi* della seguente classe sono:

- **RiepilogoCameriere(int num)**: metodo costruttore che associa il parametro in input *num* a *numerotavolo* e, inoltre, richiama il metodo *visualizza()*;
- **visualizza()**: crea la parte alta della finestra e crea la grafica relativa al pannello centrale, che conterrà l'ordine. Infine, richiama la funzione *popolaPannello()*;

- **popolaPannello()**: metodo che è in grado di popolare il pannello centrale con vari oggetti, che costituiscono l'ordinazione;
- **utilizzaPolimorfismo(Lista po)**: metodo che è in grado di utilizzare il polimorfismo per scrivere il contenuto dell'ArrayList delle ordinazioni, *listap*, sul corrispettivo file txt, che corrisponde allo scontrino del tavolo selezionato(vedi sez. 2.6).

2.3.5 CLASSE TAVOLO

Classe di back-end grazie alla quale riesco a creare un oggetto che identifica un singolo tavolo. Gli *attributi* della seguente classe sono:

- **nome**: è il nome del tavolo;
- **numero**: è il numero del tavolo;
- **stato**: è lo stato dell'ordinazione del tavolo corrente.

I *metodi* della seguente classe sono:

- **Tavolo(String name, int num, String status)**: metodo costruttore grazie al quale è possibile prendere in input tre valori, che poi vengono associati alle istanze della classe Tavolo;
- **getNome()**: metodo grazie al quale si ottiene il nome del tavolo;
- **setNome(String nome)**: metodo con il quale si può impostare il nome del tavolo;
- **getNumero()**: metodo grazie al quale si ottiene il numero del tavolo;
- **setNumero(int numero)**: metodo con il quale è possibile impostare il numero del tavolo;
- **getStato()**: metodo grazie al quale si ottiene lo stato del tavolo selezionato;
- **setStato(String stato)**: metodo grazie al quale è possibile impostare lo stato del tavolo selezionato.

2.3.6 classe TavoloSingolo

Classe di front-end che implementa la grafica del menù che visualizzerà il cameriere per prendere l'ordinazione, dopo aver selezionato un tavolo.
Gli *attributi* della seguente classe sono:

- **categorie**: Array statico di stringhe che contiene le cinque categorie fisse del menù(ANTIPASTI, PRIMI, SECONDI, CONTORNI,DOLCI);
- **listap**: oggetto di tipo *ListaPiatti*, il quale contiene tutti i piatti del menù;
- **ordinazione**: viene creato graficamente un nuovo JFrame, con il rispettivo ContentPane;
- **pannello_centrale**: viene creato come attributo anche il pannello che conterrà il menù ed i bottoni, in modo da poter agevolare il cameriere a prendere la comanda;
- **quantita**: oggetto di tipo Ordinazione, grazie al quale si riesce ad avere tutte le quantità di ogni piatto ordinato;
- **numerotavolo**: attributo di tipo intero che rappresenta il numero del tavolo su cui si sta prendendo l'ordinazione;
- **lunghezza**: costituisce la lunghezza giusta del pannello, data dalla somma della lunghezza dell'array delle categorie e la lista dei piatti ordinati;
- **contatore_quantita**: variabile che serve a far visualizzare graficamente un alert, nel caso in cui non venisse selezionata nessuna quantità.

I *metodi* della seguente classe sono:

- **TavoloSingolo(int num)**: metodo costruttore che associa il parametro in input *num* a *numerotavolo* e, inoltre, richiama il metodo *visualizza()*;
- **visualizza()**: metodo che crea la parte alta della finestra e imposta le spechifiche grafiche del pannello centrale.Inoltre, richiama il metodo *popolaPannello()* ;
- **popolaPannello()**: è in grado di popolare il pannello centrale con vari oggetti;
- **utilizzaPolimorfismo(Lista po)**: metodo che è in grado di utilizzare il polimorfismo per scrivere il contenuto dell'ArrayList *quantita* sul file *appoggio.txt* (vedi sez. 2.6).

2.4 PACKAGE CUOCO

Il package *cuoco*, che si trova all'interno del package *gestoreRistorante*, contiene tutte le classi inerenti al funzionamento del terzo bottone, correlato al ruolo del cuoco. In questo package ci sono sia la parte grafica di front-end che quella di back-end.

2.4.1 CLASSE ELENCOTAVOLICUOCO

Classe di front-end che implementa la gestione dei tavoli che visualizzerà il cuoco.
Gli *attributi* della seguente classe sono:

- **tavoli**: Array statico di stringhe che contiene i cinque tavoli fissi del ristorante(TAVOLO1, TAVOLO2, TAVOLO3, TAVOLO4, TAVOLO5);
- **listat**: oggetto di tipo *ListaTavoli*, che deve essere utilizzato per cambiare lo stato di un certo tavolo, nel caso questo venga utilizzato;
- **table_view**: viene creato graficamente un nuovo JFrame, con il rispettivo ContentPane;
- **pannello_centrale**: pannello centrale che conterrà i cinque bottoni relativi ai tavoli e il loro status;

I *metodi* della seguente classe sono:

- **ElencoTavoliCuoco()**: metodo costruttore che richiama il metodo *visualizza()*;
- **visualizza()**: crea la parte alta della finestra e setta le specifiche grafiche del pannello centrale;
- **popolaPannello()**: è in grado di popolare il pannello centrale con vari oggetti;

2.4.2 CLASSE RIEPILOGOCUOCO

Classe di front-end che implementa la grafica della comanda che visualizzerà il cuoco.
Gli *attributi* della seguente classe sono:

- **tavoli**: oggetto di tipo ListaTavoli, grazie al quale invece si riescono ad avere tutti i tavoli, con relativi attributi in un ArrayList;
- **editable_menu**: viene creato graficamente un nuovo JFrame, con il rispettivo ContentPane;

- **pannello_centrale**: viene creato come attributo anche il pannello centrale, che conterrà la comanda da evadere;
- **numerotavolo**: attributo di tipo intero che rappresenta il numero del tavolo su cui è stata presa l'ordinazione.

I *metodi* della seguente classe sono:

- **RiepilogoCuoco(int num)**: metodo costruttore che associa il parametro in input *num* all'attributo *numerotavolo*. Inoltre, richiama il metodo *visualizza()*;
- **visualizza()**: crea la parte alta della finestra e setta le spechifiche grafiche del pannello centrale;
- **popolaPannello(Scontrino comanda,JPanel pannello_variabile)**: è in grado di popolare il pannello centrale con vari oggetti;
- **utilizzaPolimorfismo(Lista po)**: metodo che è in grado di utilizzare il polimorfismo per scrivere il contenuto dell'ArrayList *tavoli* sul file *lista_tavoli.txt* (vedi sez. 2.6).

2.5 PACKAGE CASSA

Il package *cassa*, che si trova all'interno del package *gestoreRistorante*, contiene tutte le classi inerenti al funzionamento del quarto bottone correlato al ruolo del registratore di cassa. In questo package ci sono sia la parte grafica di front-end che quella di back-end.

2.5.1 CLASSE ELENCOTAVOLICASSA

Classe di front-end che implementa la gestione dei tavoli che visualizzerà la cassa.
Gli *attributi* della seguente classe sono:

- **tavoli**: Array statico di stringhe che contiene i cinque tavoli fissi del ristorante(TAVOLO1, TAVOLO2, TAVOLO3, TAVOLO4, TAVOLO5);
- **listat**: oggetto di tipo *ListaTavoli* che mantiene tutti i tavoli e i loro stati in un ArrayList, di tipo *Tavolo*;
- **table_view**: viene creato graficamente un nuovo JFrame, con il rispettivo ContentPane;
- **pannello_centrale**: pannello centrale che conterrà i cinque bottoni relativi ai tavoli e il loro status;

I *metodi* della seguente classe sono:

- **ElencoTavoliCassa()**: metodo costruttore che richiama il metodo *visualizza()*;
- **visualizza()**: crea la parte alta della finestra e setta le specifiche grafiche del pannello principale;
- **popolaPannello()**: è in grado di popolare il pannello centrale con vari oggetti.

2.5.2 CLASSE RIEPILOGOCASSA

Classe di front-end che implementa la grafica dello scontrino che visualizzerà la cassa.
Gli *attributi* della seguente classe sono:

- **tavoli**: oggetto di tipo *ListaTavoli* che mantiene tutti i tavoli e i loro stati in un ArrayList, di tipo *Tavolo*;
- **editable_menu**: viene creato graficamente un nuovo JFrame, con il rispettivo ContentPane;
- **pannello_centrale**: viene creato come attributo anche il pannello che conterrà lo scontrino, con il relativo totale;
- **numerotavolo**: attributo di tipo intero che rappresenta il numero del tavolo su cui viene effettuato lo scontrino;
- **totale**: attributo di tipo double che rappresenta il totale da pagare per il relativo tavolo.

I *metodi* della seguente classe sono:

- **RiepilogoCassa(int num)**: metodo costruttore che associa *num* a *numerotavolo* e, inoltre, richiama il metodo *visualizza()*;
- **visualizza()**: crea la parte alta della finestra e setta le specifiche grafiche del pannello principale;
- **popolaPannello(Scontrino scontrino, JPanel pannello_variabile)**: è in grado di popolare il pannello centrale con vari oggetti;
- **utilizzaPolimorfismo(Lista po)**: è in grado di utilizzare il polimorfismo per scrivere il contenuto dell'ArrayList *tavoli* sul file *lista_tavoli.txt*, in modo da aggiornare lo stato del tavolo interessato (vedi sez. 2.6).

2.5.3 CLASSE SCONTRINO

Classe back-end che contiene l'ArrayList di tutti i piatti ordinati, e gestisce vari metodi che saranno usati nella classe *RiepilogoCassa*.
Gli *attributi* della seguente classe sono:

- **listapiatti**: un ArrayList di tipo *Piatto*, contenente la lista dei piatti;
- **file1**: rappresenta lo scontrino del tavolo 1, tramite un file;
- **file2**: rappresenta lo scontrino del tavolo 2, tramite un file;
- **file3**: rappresenta lo scontrino del tavolo 3, tramite un file;
- **file4**: rappresenta lo scontrino del tavolo 4, tramite un file;
- **file5**: rappresenta lo scontrino del tavolo 5, tramite un file;
- **numerotavolo**: attributo di tipo intero che rappresenta il numero del tavolo che si è selezionato.

I *metodi* della seguente classe sono:

- **Scontrino(int numero)**: metodo costruttore che associa *numero* a *numerotavolo* e, inoltre, richiama il metodo *read()* per leggere il contenuto del file;
- **getPiatto(int indice)**: metodo grazie al quale è possibile ricavare un piatto di tipo Piatto nell'Arraylist *listapiatti*; item **size()**: metodo grazie al quale si ricava la lunghezza dell'Arraylist *listapiatti*;
- **read()**: metodo con il quale è possibile leggere dal file del tavolo interessato, in cui è contenuta la lista dei piatti ordinati per quel tavolo, e lo aggiunge all'ArrayList *listapiatti* (rappresenta il passaggio dal file alla grafica);
- **write()**: metodo che è in grado di utilizzare il polimorfismo per scrivere il contenuto dell'ArrayList *tavoli* sul file *lista_tavoli.txt* (vedi sez. 2.6).

2.6 Polimorfismo

Il polimorfismo è stato implementato in diverse classi del progetto poiché ritenevamo fosse molto utile ai fini del riutilizzo e alla pulizia del codice.
Viene utilizzato tramite l'interfaccia **Lista**, che serve per implementare il metodo **write()** per ogni tipo di oggetto che utilizza questo metodo. Il tipo di polimorfismo che è stato utilizzato in questo progetto, è quello per inclusione, ovvero è legato alle relazioni di ereditarietà tra classi, che garantisce che tali oggetti, pur di tipo differente, abbiano una stessa interfaccia: nei linguaggi ad oggetti tipizzati, le istanze di una sottoclasse possono essere utilizzate al posto di istanze della superclasse. Per fare ciò, si attua quello che è chiamato *Override dei metodi* presenti all'interno dell'interfaccia (in questo caso solamente uno), poiché permette agli oggetti appartenenti alle sottoclassi di una stessa classe, di rispondere diversamente agli stessi utilizzi. In questo caso, lo abbiamo implementato poiché il metodo da noi utilizzato scrive in modi e su file diversi.

2.7 Diagramma UML

I diagrammi UML sono molto espliciti e possono aiutare, soprattutto in un progetto sviluppato in Java, a capire meglio l'andamento degli eventi all'interno dell'applicazione.La notazione UML è semi-grafica ed è utile anche alla realizzazione di una visuale più chiara del progetto in tutta la sua completezza.

Innanzitutto, volendo partire da uno schema più lontano, vogliamo riproporre il ciclo di vita che rispetta la nostra applicazione.

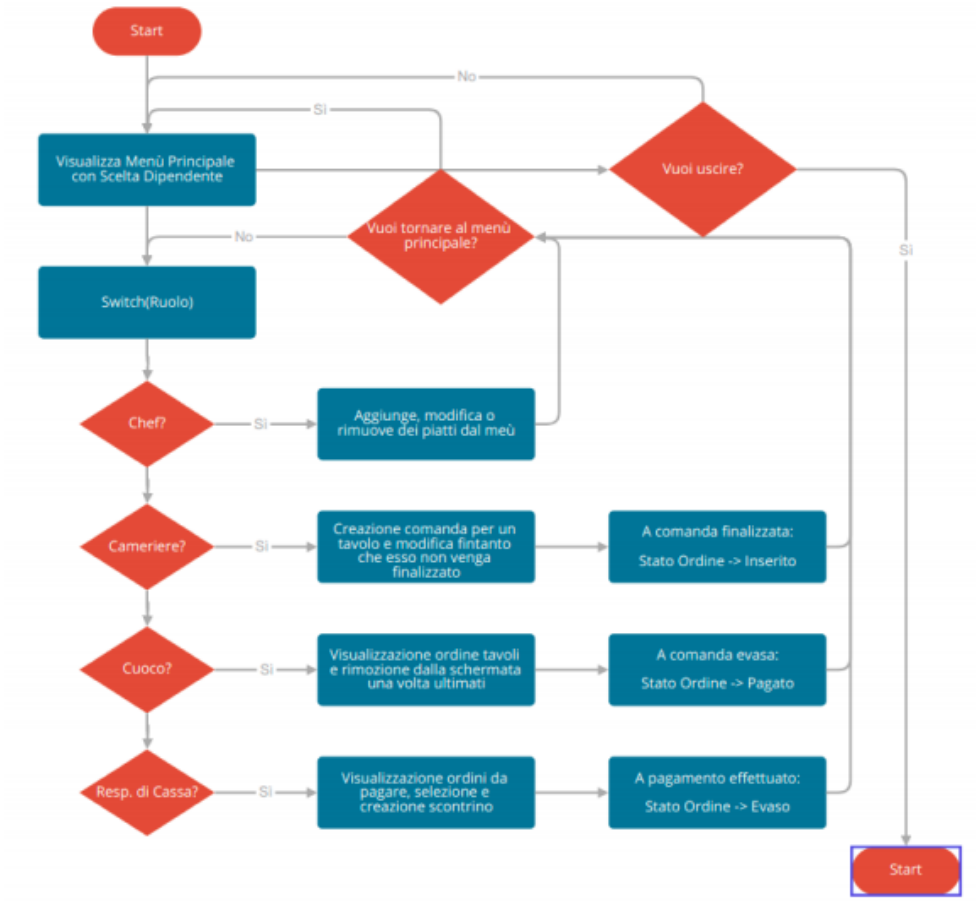


Figura 1: Questo è il diagramma generale degli eventi che possono avvenire all'interno del progetto, come richiesto.

Invece, in secondo luogo, vogliamo far vedere più da vicino la struttura del progetto secondo package e classi(che possono essere tester class, back-end class, front-end class e interfacce).
Infatti, abbiamo diviso il progetto per i ruoli richiesti, ognuno dei quali comprende poi all'interno dell'omonimo package sia la parte di front-end che di back-end.

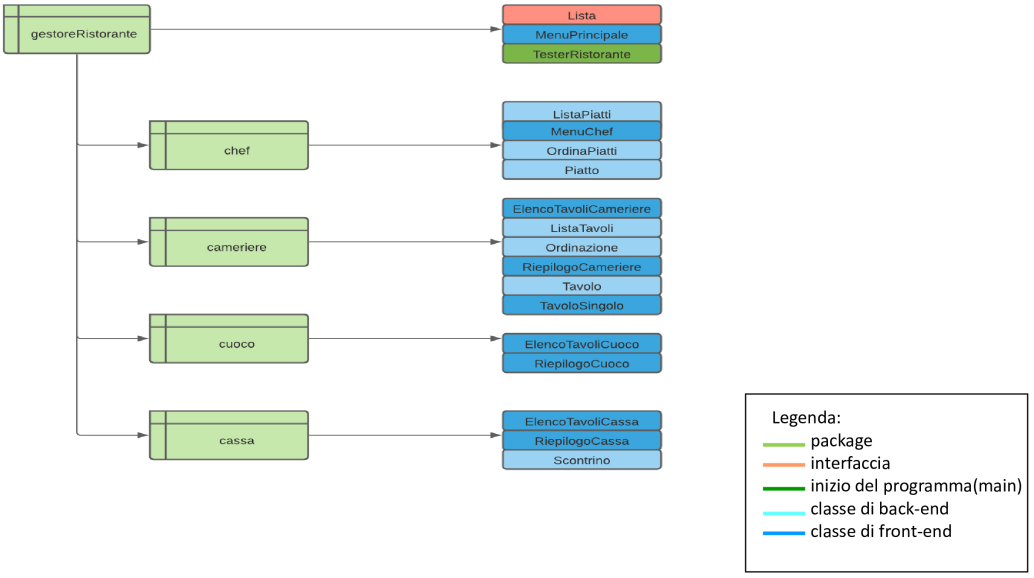


Figura 2: In questa immagine è possibile notare lo schema gerarchico che si segue grazie ai package, divisi quindi per ruolo, e ciascuno dei quali ha all'interno delle classi(di vario tipo) inerenti al package in cui sono contenute.

3

Descrizione delle Funzionalità

Questa sezione sarà dedicata alla spiegazione della progettazione del progetto, e quindi in particolar modo sarà dedicata alle scelte decisionali che sono state prese; a tal proposito, è giusto dire che sono state dedicate circa le prime due settimane alla progettazione, e solo dopo una prima visione completa, si è iniziato a scrivere codice.
La sezione sarà divisa in base ai quattro ruoli, in modo da agevolarne l'organizzazione e la lettura. Si è cercato di prendere, in generale, sempre scelte semplici e dirette, al fine di ottenere una GUI semplice da capire e usare per l'utente.

Perciò, innanzitutto, per la schermata iniziale abbiamo pensato di incolonnare quattro semplici bottoni, in modo tale che l'utente che avvia l'applicazione possa scegliere immediatamente un ruolo a scelta tra i quattro possibili; poi, per ogni bottone abbiamo deciso di far corrispondere un nuovo ciclo di eventi, di cui parleremo nelle prossime sezioni.
Inoltre, abbiamo anche deciso di ideare due tipi diversi di chiusura:

1. Con il classico pulsante **"X"** in alto a destra, si chiude l'applicazione, con l'aggiunta di salvare anche i dati raccolti fino a quel momento all'interno della sessione corrente(come ordinazioni, menù ideato dallo chef, tavoli di cui le pietanze devono ancora essere evase e pagate), quindi se l'applicazione verrà riaperta poi dopo aver cliccato "X", i dati saranno rimasti invariati, proprio come succede durante una normale giornata lavorativa in un ristorante;

2. Invece, con il pulsante in basso, **"CHIUDI SESSIONE"**, si chiude l' applicazione, e in aggiunta viene chiusa l'intera sessione corrente, come se si volesse metter fine ad un'intera giornata lavorativa (vengono di conseguenza annullate ordinazioni, tavoli da evadere e pagamenti da effettuare).

3.1 Chef

Questa sezione sarà dedicata alla spiegazione della progettazione che si cela dietro il ruolo dello chef all'interno del nostro progetto.
Per realizzare ciò che si vede tramite la GUI, abbiamo lavorato con dati strutturati, ovvero classi di back-end, che permettessero il passaggio tra la parte grafica e gli oggetti caratteristici di Java. Quindi, un passo importante è stato capire come collegare queste due cose, infatti tramite la realizzazione di alcuni metodi, utilizzati poi nella classe di front-end principale dello chef(*MenuChef*), l'implementazione funziona sia da un punto di vista grafico che funzionale.

Per il ruolo dello chef, siamo partiti dall'oggetto intorno a cui ruotava tutto : **il piatto**. Modellare l'oggetto *Piatto* è stato veramente importante, perché in questo modo ogni piatto che fa parte del menù ha delle specifiche istanze(in questo caso un nome, un prezzo e una categoria di appartenenza) e rispecchia ciò che è anche nella realtà, cioè un oggetto del menù del ristorante; per questo, gli abbiamo dedicato una classe che lo identificava.
Poi, tramite invece la classe *ListaPiatti*, abbiamo creato un ArrayList di tipo Piatto, in modo da poter mantenere tutti i piatti del menù in un'unica struttura, che però potesse anche cambiare nel tempo e con cui si potesse lavorare, tramite dei metodi, per trasferire i suoi dati all'interno sulla grafica.
Un'altra istanza importante è il file persistente dove poi le pietanze contenute in questo ArrayList vengono trascritte, in modo che questo file venga utilizzato come una specie di database, per prevenire la perdita di dati; anch'esso viene usato in alcuni metodi per essere poi letto/scritto.

```
ArrayList<Piatto> listapiatti = new ArrayList<Piatto>();
File file = new File("file/menu.txt");
```

Figura 3: In questa figura si può notare il codice grazie al quale viene creato un ArrayList di tipo Piatto e il file persistente dove sarà salvato l'intero menù, che però non sarà visibile all'utente.

Una volta creata l'entità piatto e la struttura che racchiude l'intera lista dei piatti, siamo passati alla realizzazione dei metodi, tramite i quali è possibile usare questi oggetti.
Per realizzare i metodi, abbiamo pensato alle azioni che erano legate ad un piatto e al menù, come per esempio *aggiungere, rimuovere, modificare o ricavare i dati di un piatto dal menù, leggere o ordinare il menù(per categoria)* e infine due metodi che forse sono i più importanti, cioè il metodo *read()* per leggere il contenuto del file e trascriverlo in grafica e il metodo *write()*, che fa l'azione inversa del precedente metodo.

```
public void write(){
    try {

        /**
         * Crea un oggetto BufferedWriter per scrivere l'output del file.
         */
        BufferedWriter writer = new BufferedWriter(new FileWriter(file));

        /**
        * Scrive ogni piatto all'interno del file di output.
        */
        for (Piatto datiPiatto : listapiatti){
            writer.write(datiPiatto.getName());
            writer.write("," + datiPiatto.getPrice());
            writer.write(","  + datiPiatto.getNumcategory());
            writer.newLine();
        }

        /**
         * Chiude il file.
         */
            writer.close();

    } catch(Exception ex){
        System.out.println("Exception msg: "+ex);
    }
}
```

Figura 4: In questa figura è mostrato come esempio il metodo write(), il quale legge dall'ArrayList di tipo Piatto(dalla grafica quindi) e trasferisce i dati sul file persistente, al fine di aggiornare il database, gestendo anche l'eccezione per il file.

Allo stesso modo, è stato importante anche progettare il collegamento tra questi metodi per usare l'ArrayList di piatti e la grafica principale del menù, che avviene nella classe *MenuChef*, in quanto una volta creato un oggetto di tipo *ListaPiatti* proprio nella classe *MenuChef*, si ha istantaneamente il menù di piatti pronto per essere usato nella grafica, tramite i vari metodi.
Un occhio particolare va rivolto alle azioni principali che deve svolgere lo chef: aggiungere, rimuovere o modificare un piatto; graficamente si può notare come ognuna di queste 3 azioni sia legata ad un bottone, e funzionalmente invece ai metodi creati nella classe *ListaPiatti*.Per esempio, tramite la funzione *aggiungiPiatto()*, viene appunto aggiunto un piatto al menù, che è salvato come un ArrayList di tipo Piatto, sia graficamente che funzionalmente.
NB. Quando viene aggiunto o modificato un piatto, il nome del piatto **non** deve contenere virgole; nel caso in cui, per sbaglio, l'utente immette una virgola, il programma la modifica in automatico con uno spazio.

```
ListaPiatti listap= new ListaPiatti();
```

Figura 5: Viene creato un oggetto di tipo ListaPiatti, per poterlo usare come struttura per il menù.

```
public void aggiungiPiatto(String nome_piatto, double prezzo_piatto, int category) {

    /**
     * il contenuto del frame viene pulito, e viene ricostruito; di conseguenza viene creato anche un nuovo pannello di tipo JSplitPane.
     */
    editable_menu.getContentPane().removeAll();
    contenuto= editable_menu.getContentPane();
    pannello_variabile=new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);

    /**
     * Viene creato un nuovo oggetto di tipo piatto, grazie ai 3 parametri in input.
     */
    Piatto nuovo_piatto= new Piatto(nome_piatto,prezzo_piatto, category);

    /**
     * il piatto viene aggiunto alla lista dei piatti.
     * La lista viene poi ordinata.
     * Si riscrive sul file il contenuto della lista, in questo caso con il piatto aggiunto.
     */
    listap.add(nuovo_piatto);
    listap.sort();
    utilizzaPolimorfismo(listap);

    /**
     * Viene richiamata la funzione visualizza, grazie alla quale è possibile ricreare il contenuto del frame e popolare nuovamente il pannello centrale
     * con il nuovo piatto.
     */
    visualizza();

    /**
     * Viene effettuato un refresh della finestra.
     */
    editable_menu.invalidate();
    editable_menu.validate();
    editable_menu.repaint();
}
```

Figura 6: Funzionamento della funzione utile ad aggiungere un piatto al menù.

Concludiamo questa sezione dicendo che le funzioni per modificare o rimuovere un piatto dal menù hanno dietro la stessa progettazione.

3.2 Cameriere

Questa sezione sarà dedicata alla spiegazione della progettazione che si cela dietro il ruolo di cameriere all'interno del nostro progetto.
Per il ruolo di cameriere, abbiamo pensato fosse utile creare una schermata con una lista dei tavoli, al fine di ottenere una visione completa sulla disponibilità effettiva degli stessi; questo rispecchia anche ciò che succede nella realtà, poiché un cameriere solitamente controlla se un tavolo è disponibile o meno nella sala del ristorante.
Per implementare il ruolo del cameriere, e la gestione dei tavoli, bisogna partire da un concetto base come quello del **tavolo**. Modellare l'oggetto *tavolo* è importante, in quanto questo contiene delle specifiche istanze (nome, numero e stato dell'ordinazione) che lo identificano. Poi, invece tramite la classe *ListaTavoli*, viene creato un ArrayList di tipo Tavolo, in modo da poter mantenere tutti i tavoli in un'unica struttura mutabile, che si possa manipolare tramite dei metodi per effettuare il passaggio dalla grafica al back-end e viceversa.
Un altro oggetto importante è quello del file della lista dei tavoli, grazie al quale il contenuto dell'ArrayList viene trascritto sul file, in modo tale da poterlo utilizzare come una sorta di database, nel senso che in questo file vengono salvati tutti i cambiamenti relativi allo stato di ogni tavolo.

```
ArrayList<Tavolo> listatavoli = new ArrayList<Tavolo>();
File file = new File("file/lista_tavoli.txt");
```

Figura 7: In questa figura si può notare il codice grazie al quale viene creato un ArrayList di tipo Tavolo e il file dove sarà salvato la lista dei tavoli, che però non sarà visibile all'utente.

Una volta creata l'entità tavolo e la struttura che racchiude l'intera lista dei tavoli, siamo passati alla realizzazione dei metodi, tramite i quali è possibile usare questi oggetti nella classi front-end.
Per realizzare i metodi, abbiamo pensato alle azioni che erano legate ad un tavolo, molte delle quali coincidono con quelle usate in precedenza nel ruolo di Chef. Ci possiamo focalizzare sui due metodi più importanti, ovvero *read()*, che serve a leggere il contenuto del file e trascriverlo in grafica e il metodo *write()*, che fa esattamente l'opposto del precedente metodo.

```
public void write(){
        try {

                /**
             * Crea un oggetto BufferedWriter per scrivere l'output del file.
             */
                BufferedWriter writer = new BufferedWriter(new FileWriter(file));

                /**
                * Scrive ogni piatto all'interno del file di output.
                */
                for (Tavolo datiPiatto : listatavoli){
                        writer.write(datiPiatto.getNome());
                        writer.write("," + datiPiatto.getNumero());
                    writer.write(","  + datiPiatto.getStato());
                    writer.newLine();
                }

            /**
             * Chiude il file.
             */
                writer.close();
        } catch(Exception ex){
                System.out.println("Exception msg: "+ex);
        }
```

Figura 8: In questa figura è mostrato come esempio il metodo write(), il quale legge dall'ArrayList di tipo Tavolo(dalla grafica quindi) e trasferisce i dati sul file, al fine di aggiornare il database, gestendo anche l'eccezione per il file.

Questi metodi, creati nella classe di back-end, vengono poi utilizzati nelle classi di front-end, come per esempio in *ElencoTavoliCameriere*, dove viene creato un oggetto di tipo *ListaTavoli*, grazie al quale avremo un elenco di tavoli su cui effettuare le ordinazioni. Per una scelta di progettazione, abbiamo deciso che i tavoli sono fissi e sono cinque, anche per rispecchiare la realtà, i cui un ristorante ha un numero fisso di tavoli.

```
String nomitavoli[] ={"TAVOLO 1", "TAVOLO 2", "TAVOLO 3", "TAVOLO 4","TAVOLO 5"};
ListaTavoli listat = new ListaTavoli();
```

Figura 9: Vengono creati un Array statico, che contiene i nomi dei tavoli, e un oggetto di tipo ListaTavoli, per poterlo usare invece come struttura dati.

Una volta scelto un tavolo, si apre una nuova finestra tramite la quale il cameriere può effettuare l'ordinazione per il tavolo scelto. Da un punto di vista grafico questo viene implementato tramite la classe *TavoloSingolo*, che grazie a dei bottoni permette di scegliere la quantità dei piatti desiderati; invece, da un punto di vista funzionale questo avviene tramite la classe *Ordinazione*. Per entrambe le classi, è possibile riscontrare delle somiglianze con le classi *ListaPiatti* e *MenuChef* del ruolo Chef, ma differiscono per funzionalità, in particolare la classe *Ordinazione* utilizza un nuovo file, *appoggio.txt*, essenziale per il salvataggio temporaneo dell'ordinazione che si sta facendo in quel momento.

```
File file = new File("file/appoggio.txt");
```

Figura 10: Viene creata una variabile di tipo File, in cui viene salvato il contenuto del file *appoggio.txt*.

Il file di appoggio serve per salvare momentaneamente l'ordinazione che è appena stata effettuata, in quanto l'ordinazione può essere cambiata finché non viene confermata nella fase di riepilogo.
Una volta scelte le quantità, è possibile visualizzare l'ordinazione provvisoria nella schermata finale di riepilogo e, una volta ricontrollata, farla diventare un'ordinazione effettiva.
A questo punto, il contenuto del file di appoggio viene copiato all'interno del file relativo al tavolo su cui si è presa l'ordinazione (che sarà poi lo scontrino finale); copiato il contenuto del file di appoggio, quest'ultimo viene eliminato e poi ricreato.

```
     */
    FileInputStream instream = null;
    FileOutputStream outstream = null;
    try{

            /**
             * In base al numero del tavolo si copia l'ordine nel ri
             */
        instream = new FileInputStream(infile);

        if (numerotavolo==0) {
            File outfile =new File("file/scontrino_tavolo1.txt");
            outstream = new FileOutputStream(outfile);
        }
        else if (numerotavolo==1) {
            File outfile =new File("file/scontrino_tavolo2.txt");
            outstream = new FileOutputStream(outfile);
        }
        else if (numerotavolo==2) {
            File outfile =new File("file/scontrino_tavolo3.txt");
            outstream = new FileOutputStream(outfile);
        }
        else if (numerotavolo==3) {
            File outfile =new File("file/scontrino_tavolo4.txt");
            outstream = new FileOutputStream(outfile);
        }
        else {
            File outfile =new File("file/scontrino_tavolo5.txt");
            outstream = new FileOutputStream(outfile);
        }

        byte[] buffer = new byte[1024];

        int length;
        while ((length = instream.read(buffer)) > 0){
            outstream.write(buffer, 0, length);
        }

        instream.close();
        outstream.close();
    }catch(IOException ioe){
            ioe.printStackTrace();
     }
if(infile.exists()){
        infile.delete();
    }
    try {
                        infile.createNewFile();
                } catch (IOException e) {
                        e.printStackTrace();
                }
```

Figura 11: In base al tavolo scelto, viene scritto il file con il contenuto del file di appoggio.

3.3 Cuoco

Questa sezione sarà dedicata alla spiegazione della progettazione che si cela dietro il ruolo di cuoco all'interno del nostro progetto.
Per il ruolo di cuoco, molte funzionalità di back-end vengono riprese da cameriere e le differenze principali avvengono soprattutto da un punto di vista grafico(front-end); anche i nomi delle classi sono simili, infatti la classe relativa alla gestione dei tavoli è chiamata *ElencoTavoliCuoco*, che differisce da *ElencoTavoliCameriere* per un'unica funzionalità: un determinato tavolo è accessibile solo se lo stato non è **NI** oppure **E**, e quindi se è stata effettuata un'ordinazione sullo stesso.

```
if (listat.getTavolo(i).getStato().equals("NI") || listat.getTavolo(i).getStato().equals("E")) {
    Tavolo.setEnabled(false);
}
```

Figura 12: Controllo grazie al quale è possibile verificare se un certo tavolo è accessibile o meno.

Dal momento in cui un tavolo è accessibile, è possibile visualizzare l'ordinazione nella schermata finale di riepilogo, in cui il cuoco potrà evadere l'ordine con i relativi piatti. Una volta fatto ciò, lo stato dell'ordine relativo a quel tavolo cambia da **I** a **E**, in modo da poter essere poi visibile nel ruolo successivo, cassa.

3.4 Cassa

Questa sezione sarà dedicata alla spiegazione della progettazione che si cela dietro il ruolo di cassa all'interno del nostro progetto.
Per il ruolo di cassa, molte funzionalità di back-end vengono riprese da cuoco e le differenze principali avvengono soprattutto da un punto di vista grafico(front-end); anche i nomi delle classi sono simili, infatti la classe relativa alla gestione dei tavoli è chiamata *ElencoTavoliCassa*, che differisce da *ElencoTavoliCuoco* per un'unica funzionalità: è possibile effettuare il pagamento per un determinato tavolo solo se lo stato non è **NI** oppure **I**, e quindi se è stata effettuata un'ordinazione sullo stesso e poi evasa.

```
if (listat.getTavolo(i).getStato().equals("NI")||listat.getTavolo(i).getStato().equals("I")) {
    Tavolo.setEnabled(false);
}
```

Figura 13: Controllo grazie al quale è possibile verificare se su un certo tavolo è possibile effettuare il pagamento.

Una volta scelto un tavolo è possibile visualizzare lo scontrino finale relativo allo stesso; questo avviene modellando una nuova classe di back-end *Scontrino*, che ci permette di accedere al file giusto che contiene tutti i dati relativi all'ordine da pagare, conoscendo il numero del tavolo su cui stiamo operando. In scontrino, uno dei metodi più importanti è il metodo *read()*, poiché ci permette di accedere al file desiderato e copiare il contenuto dello stesso in un ArrayList di tipo *Piatto*, che verrà utilizzato poi dalla classe di front-end *RiepilogoCassa*.
Qui di seguito si inserisce uno spezzone di tale metodo:

```java
public void read() {
    try {

        /**
         * Crea un oggetto BufferedReader per leggere l'input del file.
         */

        if (numerotavolo==0) {
            BufferedReader reader = new BufferedReader(new FileReader(file1));
            String currentLine = reader.readLine();
            while (currentLine != null){
                String[] datiPiatto = currentLine.split(",");

                String Name = datiPiatto[0];
                String Price =datiPiatto[1];
                String NumCat= datiPiatto[2];
                int numint= Integer.parseInt(NumCat);
                double prezzo = Double.parseDouble(Price);
                /**
                * Crea un oggetto Piatto e lo aggiunge all'ArrayList.
                */
                listapiatti.add(new Piatto( Name, prezzo, numint));

                currentLine = reader.readLine();
            }
            reader.close();
```

Figura 14: parte del metodo read().

Una volta visualizzato lo scontrino e pagato l'importo totale, lo stato del relativo tavolo viene cambiato da *E* a *NI*, cioè il tavolo è di nuovo disponibile e quindi è possibile effettuare una nuova ordinazione.

4

Manuale della GUI

Il progetto è stato ideato e progettato con una parte grafica(GUI) e, proprio in questa sezione, verrà spiegato il significato e l'utilità di ogni schermata insieme ad una breve descrizione di ogni elemento contenuto al suo interno.

4.1 Menù Principale

La prima interfaccia che si incontra è il menù di partenza, dove è possibile selezionare i vari ruoli, tramite dei bottoni.

Figura 15: Ci sono due tipi di chiusure; se si usa il pulsante *"CHIUDI SESSIONE"* in basso il programma verrà chiuso e verranno puliti tutti i file che contengono i dati di ordinazione, scontrini e gli stati dei tavoli torneranno al loro valore di partenza(*NI*). La seconda chiusura, è la tipica "X" in alto a destra, con la quale si potrà chiudere la finestra, non eliminando i dati all'interno del programma.

4.2 Chef

Cliccando il bottone **CHEF**, si trova il menù che potrà creare e modificare lo chef del ristorante; inoltre, all'interno della finestra, sono presenti tre bottoni: una freccia in alto(<-), con la quale si potrà tornare al menù principale, un più(+) per aggiungere un piatto al menù, selezionando poi la categoria (da un menù a tendina), inserendo il nome e il prezzo che si desidera.

Infine, è presente anche la possibilità di modificare un piatto tramite il bottone a forma di penna.

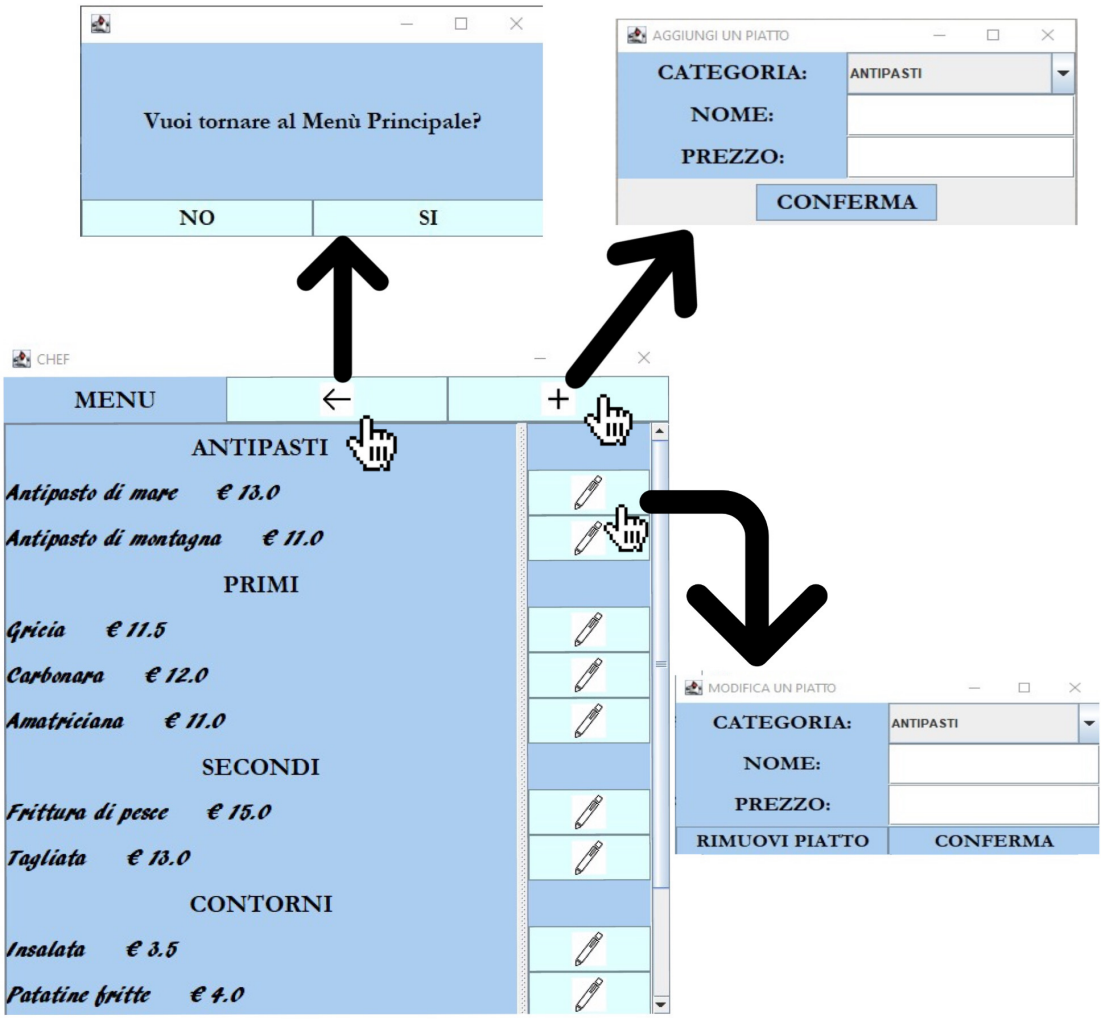


Figura 16: La prima freccia mostra la possibilità di tornare indietro, la seconda mostra la finestra per aggiungere un piatto e la terza freccia per modificarlo.

4.2.1 Alert Chef

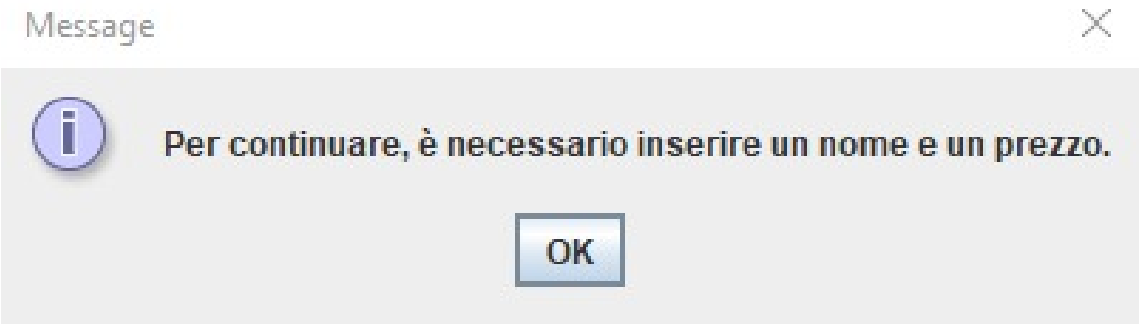


Figura 17: Nel momento in cui si clicca conferma durante l'aggiunta o la modifica di un piatto e non vengono inseriti sia un nome che un prezzo, questo genererà un alert, come mostrato in figura.

4.3 Cameriere

Cliccando invece il bottone *Cameriere*, troviamo la lista dei tavoli con il loro stato corrispondente a destra, che può essere "NI" ovvero "non inserito" oppure "I", "inserito". Scelto il tavolo da cui prendere l'ordinazione, ci si ritrova quindi il menù(ideato precedentemente dallo chef) con la possibilità di aggiungere e togliere le pietanze con un quantificatore fatto con due bottoni, "+" e "-".
Cliccando poi il tasto *conferma*, si passa alla pagina di riepilogo dell'ordine, con il bottone finale "INSERISCI ORDINE" che invierà l'ordine verso il ruolo del cuoco.

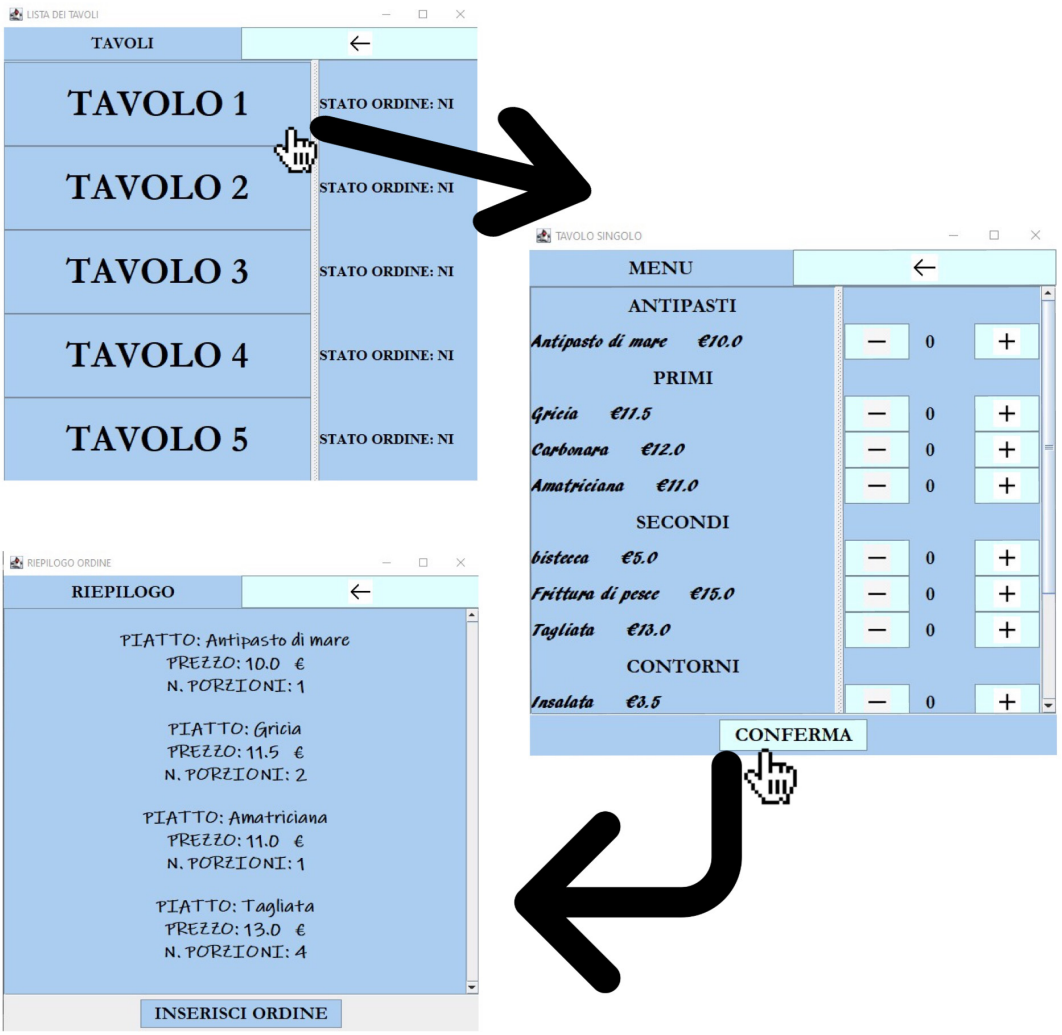


Figura 18: La prima freccia porta al menù con le pietanze ordinabili, la seconda mostra la finestra di riepilogo con la possibilità di tornare indietro o di inserire l'ordine. La possibilità di tornare indietro è sempre disponibile grazie alla freccia posta in altro a destra.
Nel caso in cui venga cliccato il pulsante *back*(<-) nella finestra di riepilogo dell'ordine, quest'ultimo viene azzerato.

4.3.1 Alert Cameriere

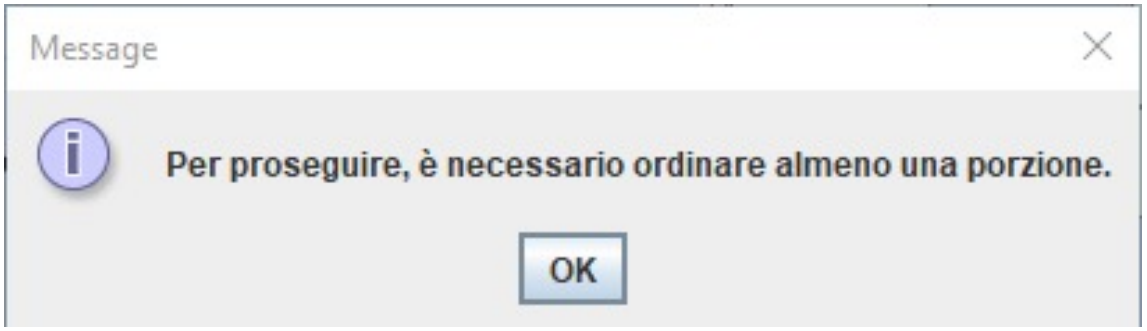


Figura 19: Durante l'ordinazione, se non viene inserito nessun piatto e si procede con conferma, arriverà un alert di questo tipo, in quanto non è possibile effettuare un ordine vuoto.

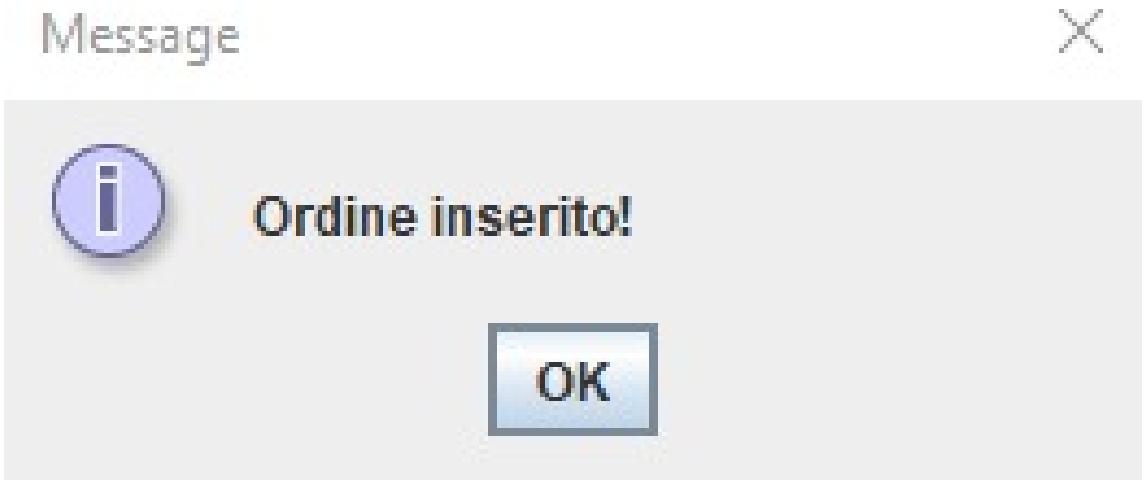


Figura 20: Nella pagina finale del riepilogo dell'ordine, se si clicca *INSERISCI ORDINE* si avrà un alert come questo qui in alto, che indica che l'ordine è andato a buon fine.

4.4 Cuoco

Cliccando il tasto *CUOCO* troviamo, come per cameriere, la lista dei tavoli, dove il bottone relativo ad un certo tavolo sarà cliccabile solo se l'ordine è stato inserito dal cameriere. Il tavolo cliccabile avrà come stato "I" ovvero "inserito" altrimenti "NI".
Cliccando il tavolo disponibile, il cuoco visualizzerà l'ordine inviato dal cameriere con le relative quantità (l'intera comanda). Quando il cuoco avrà concluso ed evaso i piatti del tavolo cliccando "EVADI ORDINE", l'ordine sarà pronto per lo step successivo, "CASSA".

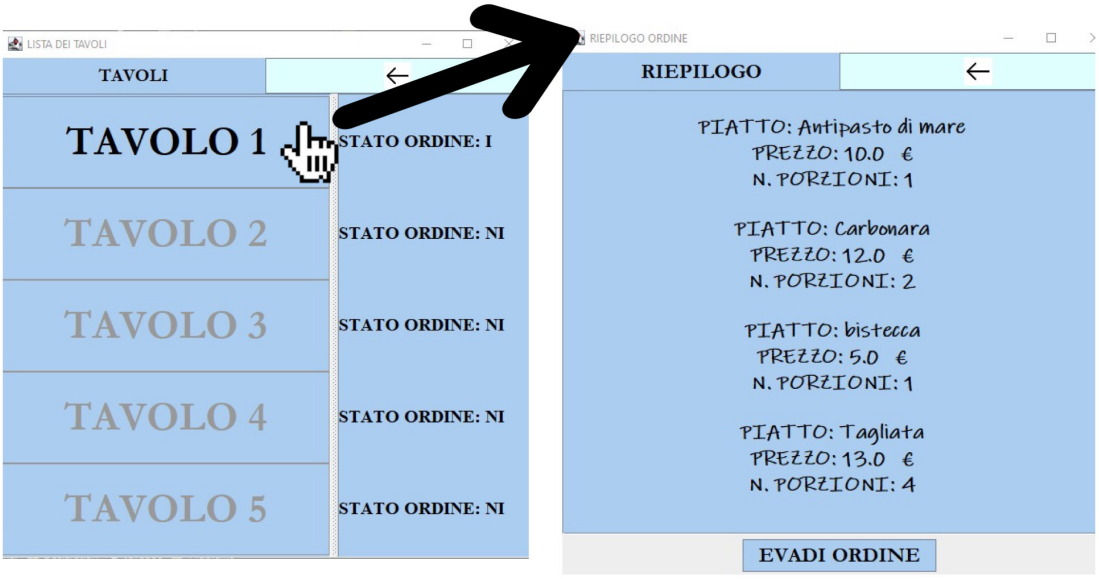


Figura 21: La possibilità di tornare indietro è sempre disponibile grazie alla freccia posta in altro a destra.

4.4.1 Alert Cuoco

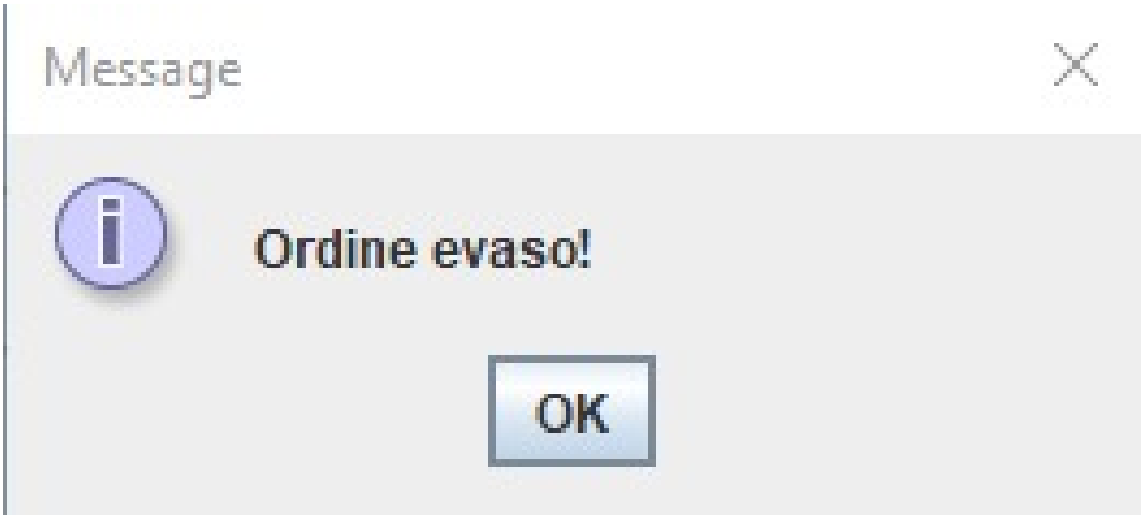


Figura 22: Nella pagina finale del riepilogo dell'ordine completato, ovvero quando tutti i piatti sono stati preparati, quando si cliccherà "EVADI ORDINE" avremo un alert come questo qui in alto.

4.5 Cassa

L'ultimo ruolo cliccabile è "CASSA". La prima finestra sarà la lista dei pulsanti "TAVOLO" dove saranno cliccabili solo quelli evasi dal ruolo precedente, "CUOCO". Questi avranno come stato "E" ovvero "evaso", altrimenti "NI". Cliccando il tavolo in questione visualizzeremo l'intero ordine con i relativi prezzi e quantità. A fondo pagina troviamo il prezzo totale con il pulsante "PAGA ORDINE" che se cliccato ci riporterà alla pagina precedente cambiando lo stato del tavolo in questione.

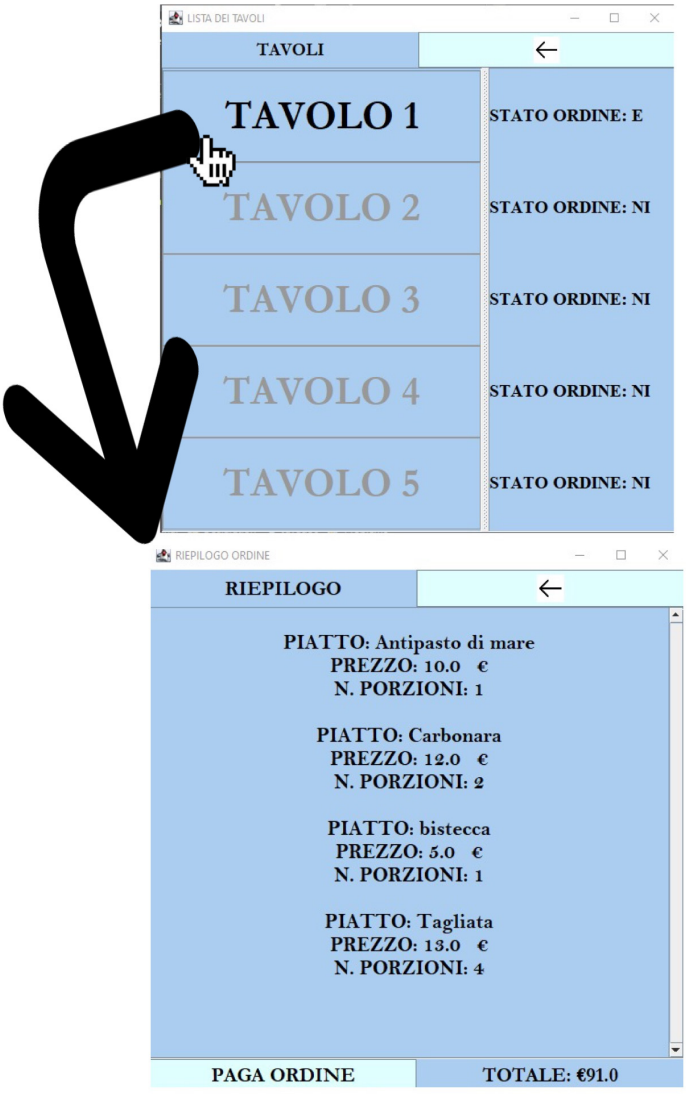


Figura 23: La possibilità di tornare indietro è sempre disponibile grazie alla freccia posta in altro a destra.

4.5.1 Alert Cassa

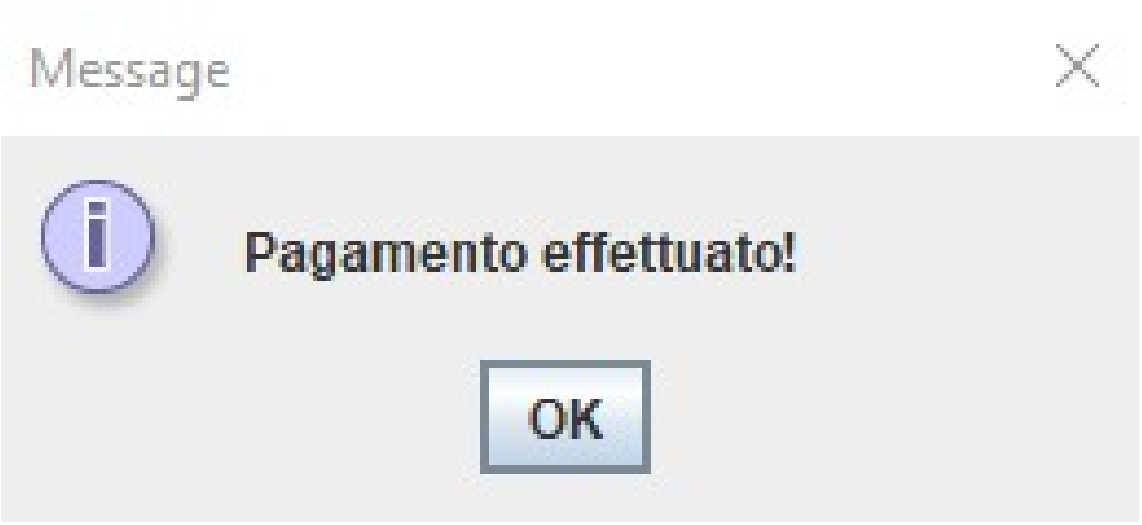


Figura 24: Nella pagina finale del riepilogo dell'ordine completato, ovvero quando l'intero ordine è stato evaso, quando si cliccherà "PAGA ORDINE" avremo un alert come questo qui in alto.

5

Referenti di Sviluppo

Per questa sezione, abbiamo deciso di dividere la sezione in tre sottosezioni, ciascuna per ogni componente del gruppo, in modo da vedere in modo chiaro e ordinato le suddivisioni del lavoro per il progetto.

5.1 Alessia Ciarla

Si è occupata principalmente della schermata di partenza dell'applicativo, più precisamente delle classi *TesterRistorante e MenuPrincipale*, e del ruolo dello chef, con le classi *MenuChef* e in piccola parte con le classi *ListaPiatti* e *Piatto*.
Inoltre, ha aiutato nella scrittura di codice, in particolar modo nei metodi, del ruolo del *cameriere*.

5.2 Daniele Fasano

Si è occupato principalmente dei ruoli cameriere, cuoco e cassa, e in piccola parte del ruolo chef.
Nel ruolo dello chef con le classi *ListaPiatti* e *OrdinaPiatti*, mentre per gli altri tre ruoli ha implementato circa tutte le classi con l'aiuto in contemporanea degli altri due membri del gruppo. Inoltre, ha implementato l'interfaccia Lista, che viene poi implementata nei vari ruoli all'interno del programma.

5.3 Patrizio Germani

Si è occupato in piccola parte del ruolo dello chef, e nei ruoli cameriere, cuoco e cassa.
Nel ruolo dello chef con la classe *Piatto*, mentre per gli altri tre ruoli ha contribuito all'implementazione circa di tutte le classi con l'aiuto in contemporanea degli altri due membri del gruppo.

6

README

Abbiamo deciso di lasciare per iscritto anche nel report le informazioni racchiuse nel README, le quali sono utili alla compilazione del progetto ed alla sua esecuzione, nel caso in cui il file README.txt andasse perso.

- Per quanto riguarda il sistema operativo, l'applicativo che abbiamo realizzato è funzionante sia in Windows che in macOS, ma è consigliato in Windows, in quanto si riesce a visualizzare una grafica migliore;
- Inoltre, per la codifica, al fine di riuscire a visualizzare i simboli dell'applicazione nel modo corretto(ad esempio il simbolo dell'euro, "€"), è necessario impostare il Text File Encoding in UTF-8;
- Per visualizzare la schermata d'inizio, è necessario aprire la classe TesterRistorante, e far partire il programma da lì;
- Per concludere, la versione di Java utilizzata per il progetto è la 8.0.2810.9 .

Per questioni di Copyright, l'ultima modifica aggiornata del progetto è datata Giugno/Luglio 2021 e lasciamo qui i contatti dei membri del gruppo:

1. Alessia Ciarla 1814975, <ciarla.1814975@studenti.uniroma1.it>;
2. Patrizio Germani 1793486, <germani.1793486@studenti.uniroma1.it>;
3. Daniele Fasano 1815182, <fasano.1815182@studenti.uniroma1.it>.