Indice

PrestoManager - Semplice gestore di ristorante

Valerio Colella

Introduzione

PrestoManager è un programma che permette di gestire un ristorante separando le manzioni di Chef, Cameriere, Cuoco e Cassa, secondo le specifiche del progetto "Gestore di un Ristorante".
Nonostante fosse consigliato un gruppo di 3 persone per completare il progetto, non riuscendo a formare un gruppo ho deciso di continuare da solo.
Il progetto è inizialmente sviluppato usando il modello **Model - View - Controller**.
Per come è fatto Swing le parti View e Controller si trovano nella stessa classe ma sono separate internamente: tipicamente la parte View si trova nel costruttore, mentre la parte Controller è costituita principalmente da `Action` definite in `initActions()` e `initPostActions()`. Vedi Page

1 Descrizione delle classi

Nota: il logger è escluso dagli attributi delle classi. Viene usato SLF4J con implementazione SimpleLogger.
Se non fanno nulla di particolare, getter e setter sono abbreviati come *(g)*, *(s)* o *(g/s)* accanto agli attributi collegati

Legenda:

○ attributo pubblico	● metodo pubblico
◇ attributo protected	◆ metodo protected
□ attributo private	■ metodo private

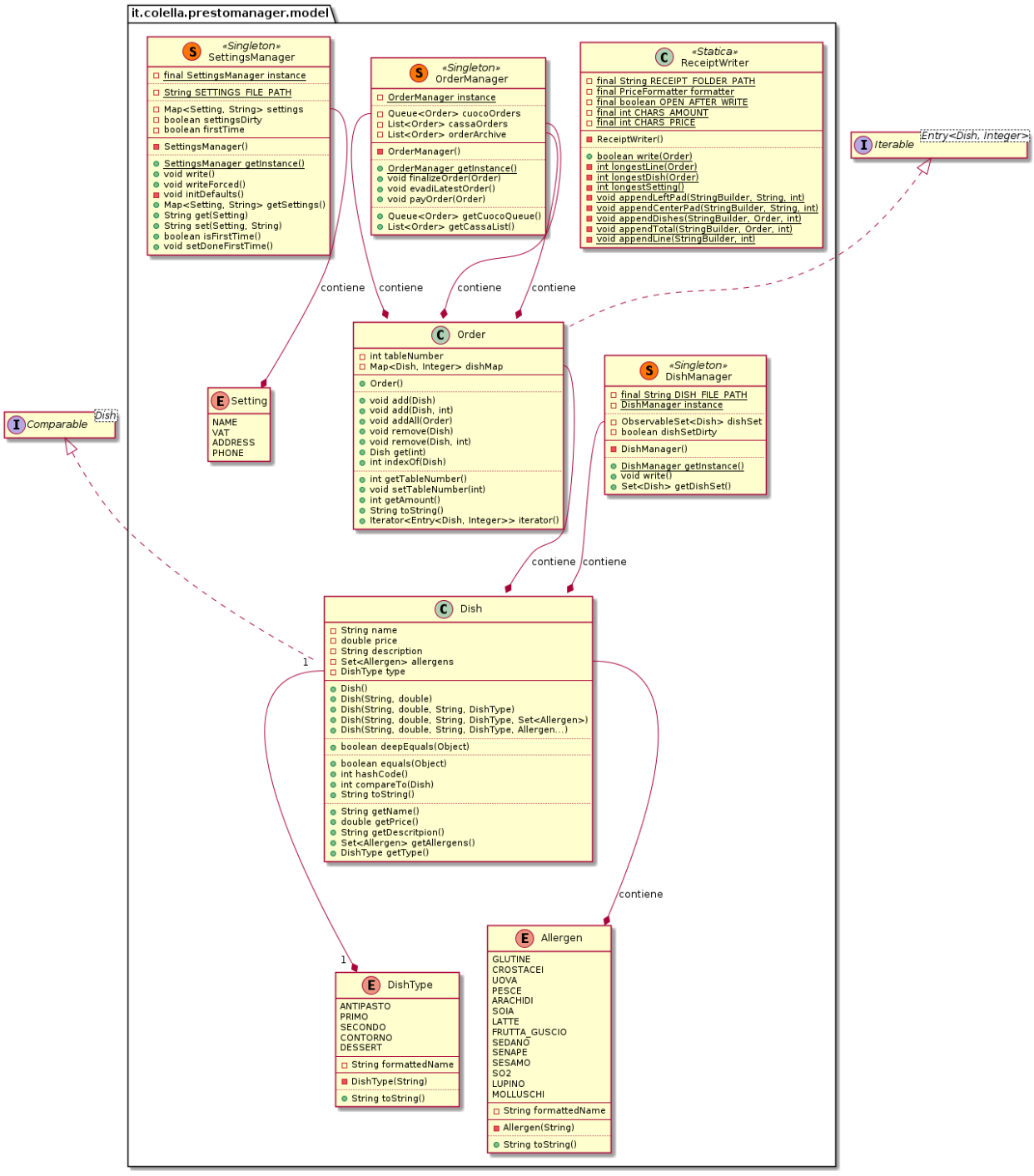
1.1 PrestoManager

La classe principale del progetto, contiene il metodo `main`, nel quale:

- Imposta la lingua, usato tra l'altro per stampare i prezzi in stile italiano - "5.000,00" anziché "5000.00"
- Legge da disco piatti e impostazioni (delegato a DishManager e SettingsManager)
- Inizializza FlatLaf
- Crea un JFrame e ne imposta le dimensioni e l'icona
- Aggiunge un WindowListener al JFrame in modo che, quando l'utente tenta di uscire:
  - Se non ci sono ordini ancora da evadere o pagare, salva piatti e impostazioni ed esce
  - Se ci sono ordini ancora da evadere o pagare, apre un popup di conferma. In base alla scelta dell'utente può annullare la chiusura del JFrame oppure salvare piatti e impostazioni e uscire
- Imposta la pagina corrente a MenuPage
- Rende il JFrame visibile

1.2 Model

Descrizione delle classi del Model dell'applicazione, cioè i dati su cui si basa la logica dell'applicazione



1.2.1 DishType e Allergen

DishType è un `enum` che descrive il tipo di un piatto (bevanda, antipasto, primo, secondo, contorno, dessert). È facilmente espandibile, in quanto semplicemente aggiungendo un'istanza a DishType la GUI si adatta automaticamente.

Allergen è un `enum` che rappresenta un allergene che può essere presente in un piatto (glutine, crostacei, uova, pesce, arachidi, soia, latte, frutta a guscio, sedano, senape, semi di sesamo, anidride solforosa o solfiti, lupino, molluschi), secondo il REG. (UE) n. 1169/2011
Sia **DishType** che **Allergen** hanno i seguenti metodi e attributi:

- `String formattedName` *(g)* - Nome formattato dell'enum
- `DishType(String formattedName)` e `Allergen(String formattedName)` - costruttori, impostano formattedName
- `@Override String toString()` - override per poter usare il nome formattato come parte della GUI. Non sovrascrivere `toString` e invece fornire un getter avrebbe portato a dover creare una classe Renderer a parte per entrambe le classi per poterle usare nella GUI

1.2.2 Dish

Rappresenta un piatto. Implementa *Comparable⟨Dish⟩*. Attributi:

- `String name` *(g)* - nome
- `double price` *(g)* - prezzo
- `String description` *(g)* - descrizione
- `Set<Allergen> allergens` *(getter restituisce un* unmodifiableSet*)* - set di allergeni contenuti nel piatto
- `DishType` type *(g)* - tipo del piatto

Metodi:

- `Dish()` - crea un piatto con dati di default
- `Dish(String name)` - crea un piatto con dati di default e un nome dato
- `Dish(String name, double price, String description)` - crea un piatto dai dati forniti con nessun allergene
- `Dish(String name, double price, String description, DishType type, Set<Allergen> allergens)` - crea un piatto dai dati forniti
- `Dish(String name, double price, String description, DishType type, Allergen... allergens)` - crea un piatto dai dati forniti
- `@Override boolean equals(Object other)` e `public int hashCode()` - sovrascritti in modo che due piatti siano uguali $\iff$ i nomi dei due piatti, ignorata capitalizzazione e spazi iniziali e finali, sono uguali
- `@Override int compareTo(Dish arg0)` - per ordinare i piatti per nome
- `boolean deepEquals(Dish other)` - determina se due piatti sono esattamente uguali tra loro, usato in DishPage per verificare se l'utente quando preme il tasto "Annulla" ha modificato il piatto e nel caso avvertirlo.

  Inizialmente è stato scritto usando la **Riflessione**, iterando cioè su tutti gli attributi di entrambi i piatti e confrontandoli.

  Considerata la lentezza della riflessione, ed il fatto che in futuro si potrebbe aggiungere un attributo del piatto non direttamente modificabile dall'utente (del quale quindi non interessa se è esattamente uguale a quello dell'altro piatto), la versione corrente verifica direttamente se nome, prezzo, descrizione, allergeni e tipo sono esattamente uguali per entrambi i piatti.

  La versione originale è mantenuta come commento.
- `@Override String toString()` - stampa il piatto come `Dish(nome|prezzo|tipo)`

1.2.3 Order

Rappresenta l'ordine di un tavolo. Un tavolo potrà fare altri ordini in seguito, quindi va immaginato più come *comanda* che come *conto*. Attributi:

- `Map<Dish, Integer> dishMap` *(getter restituisce un* unmodifiableSet*)* - mappa **ordinata** di tipo { piatto : numero porzioni }
- `int tableNumber` *(g/s)* - numero del tavolo che ha fatto l'ordine. Mai ≤ 0

Metodi:

- `Order()` - inizializza gli attributi
- `void add(Dish d)` - aggiunge una porzione di un piatto all'ordine
- `void remove(Dish d)` - rimuove una porzione di un piatto dall'ordine
- `void add(Dish d, int amount)` - aggiunge multiple porzioni di un piatto all'ordine
- `void remove(Dish d, int amount)` - rimuove multiple porzioni di un piatto dall'ordine. Se l'ordine non contiene quel piatto, NOP
- `void addAll(Order o)` - aggiunge tutte le porzioni di tutti i piatti di un altro ordine a questo
- `int getAmount(Dish d)` - restituisce quante porzioni di un piatto sono state ordinate
- `double calculateTotal()` - calcola e restituisce il prezzo totale per quest'ordine
- `Dish get(int index)` - restituisce il piatto all'indice `index` di quest'ordine
- `int indexOf(Dish d)` - restituisce l'indice sel piatto `d` in quest'ordine
- `@Override Iterator<Entry<Dish, Integer>> iterator()` - iteratore sull'ordine
- `@Override String toString()` - stampa l'ordine come `Order(MAPPA)tavolo=TAVOLO`

1.2.4 DishManager

Classe **singleton** responsabile della lettura e scrittura di piatti su disco fisso. Attributi:

- `static final DishManager instance` - essendo un singleton, unica istanza della classe
- `static final DISH_FILE_PATH` - percorso (relativo) del file per la lettura e scrittura dei piatti
- `Set<Dish> dishSet` *(getter restituisce un* unmodifiableSet*)* - Set contenente tutti i piatti disponibili nel programma, tipicamente letti da file all'avvio
- `boolean dishSetDirty` - segna se il set in memoria volatile è al momento diverso da quello su disco fisso

Metodi:

- `DishManager()` - legge i piatti da disco, se il file esiste. Altrimenti inizializza valori di default
- `static DishManager getInstance()` - restituisce l'unica istanza della classe
- `void write()` - salva i piatti su disco fisso, se necessario (cioè se `dishSetDirty == true`
- `void writeForced()` - salva i piatti su disco fisso per forza
- `boolean add(Dish d)` - Aggiunge `d` a `dishSet` e imposta `dishSetDirty` a `true`. Restituisce `true` se il set non conteneva un piatto `x` tale che `x.equals(d)`, altrimenti `false`
- `boolean remove(Dish d)` - Rimuove `d` da `dishSet` e imposta `dishSetDirty` a `true`. Restituisce `true` se il set conteneva un piatto `x` tale che `x.equals(d)`, altrimenti `false`

1.2.5 OrderManager

Classe **singleton** responsabile del corretto passaggio di ordini tra cameriere, cuoco e cassa. Attributi:

- `static final OrderManager instance` - unica istanza della classe
- `final List<Order> cuocoOrders` *(getter restituisce un* unmodifiableList*)* - lista degli ordini assegnati al cuoco
- `final Map<Integer, Order> cassaOrders` *(getter restituisce un* unmodifiableMap*)* - mappa { tavolo : ordine } degli ordini evasi, assegnati quindi alla cassa
- `final List<Order>` orderArchive - lista degli ordini pagati. Al momento la lista non ha utilità, ma è pensata per poter facilmente analizzare gli ordini pagati a fine giornata o generare statistiche

Metodi:

- `OrderManager()` - inizializza gli attributi
- `OrderManager getInstance()` - restituisce l'unica istanza della classe
- `void finalizeOrder(Order order)` - finalizza un ordine. Non potrà più essere modificato e diventa visibile al cuoco
- `void evadi(int index)` - evade l'ordine all'indice dato
- `void payTable(int tableNum)` - paga il conto del tavolo `tableNum`. Non sono effettuati controlli per determinare se il tavolo può pagare (vedi `isPayable(int)` o meno)
- `boolean isPayable(Order o)` - Restituisce `true` se l'ordine è stato evaso ma è da pagare, e non ci sono ordini dello stesso tavolo che devono essere ancora evasi. Altrimenti `false`
- `boolean isPayable(int tableNum)` - Restituisce `true` se il tavolo ha ordini evasi ma da pagare, e non ci sono ordini dello stesso tavolo che devono essere ancora evasi. Altrimenti `false`
- `Map<Integer, Order> getPayableOrders()` - restituisce una mappa read-only degli ordini **pagabili**, cioè quelli già evasi ma da pagare, di tavoli senza ordini ancora da evadere
- `Integer[] getPayableTables()` - restituisce i tavoli **pagabili**, cioè quelli che hanno tutti gli ordini già evasi

1.2.6 Setting

Enum che rappresenta il tipo di impostazioni che possiamo leggere e salvare su disco (NAME, VAT, ADDRESS, PHONE).

1.2.7 SettingsManager

Classe **singleton** responsabile della lettura e scrittura delle impostazioni su disco fisso. Attributi:

- `static final SettingsManager instance` - unica istanza della classe
- `static final SETTINGS_FILE_PATH` - percorso (relativo) del file per la lettura e scrittura delle impostazioni
- `boolean settingsDirty` - segna se la mappa in memoria volatile è al momento diversa da quella su disco fisso
- `boolean firstTime` - segna se è la prima volta che si apre il programma (o dopo un reset)

1.2.8 ReceiptWriter

Classe puramente statica per stampare lo scontrino a file di testo. Attributi:

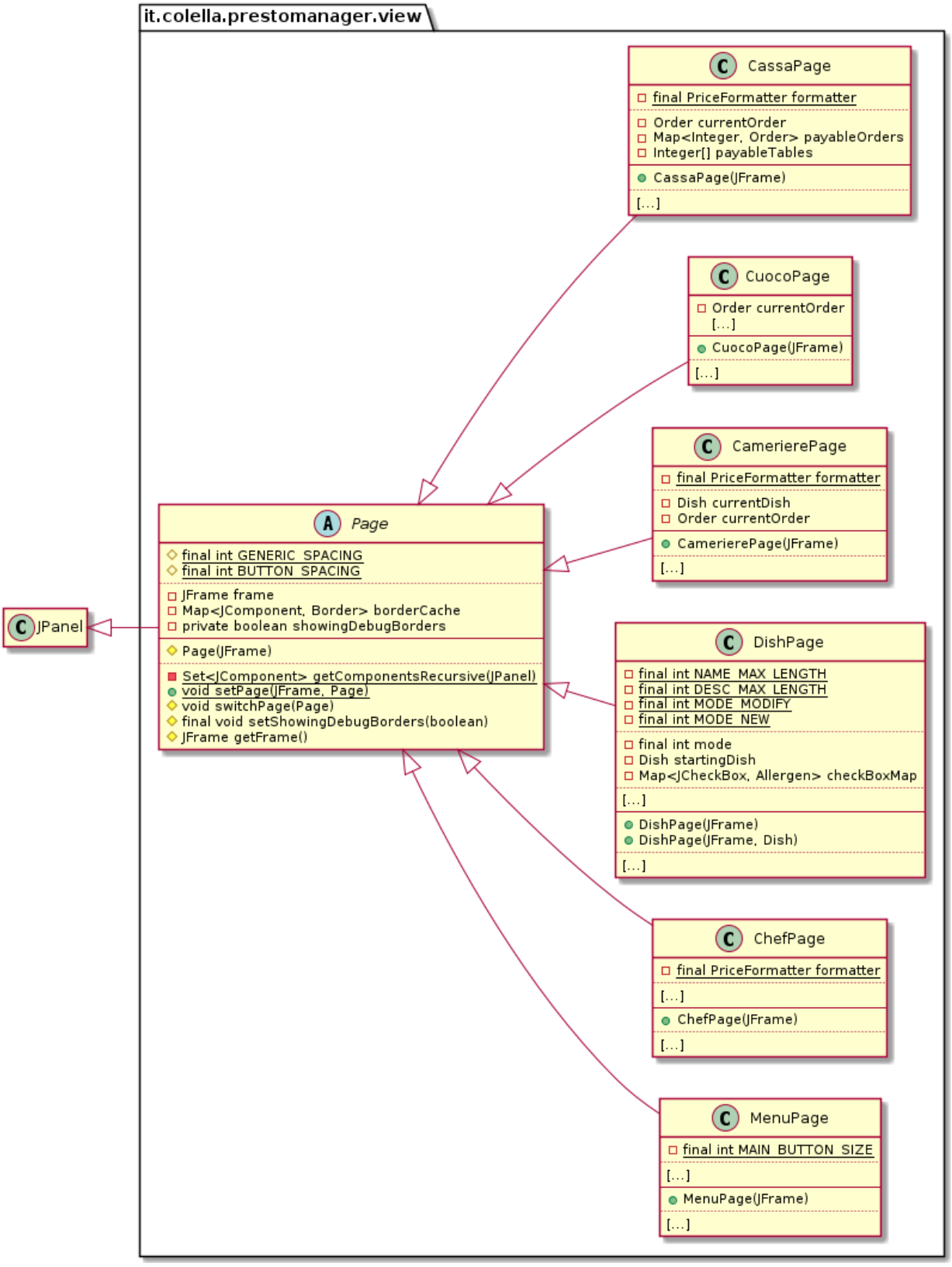
- `static final String RECEIPT_FOLDER_PATH` - percorso (relativo) degli scontrini
- `static final PriceFormatter formatter` - necessario per formattare il prezzo
- `static final boolean OPEN_AFTER_WRITE` - se aprire o meno il file con lo scontrino dopo la scrittura, deciso alla compilazione
- `static final int CHARS_AMOUNT` - numero di caratteri necessari per stampare il numero di porzioni di un piatto
- `static final int CHARS_PRICE` - numero di caratteri necessari per stampare il prezzo

Metodi:

- `static boolean write(Order order)` - scrive lo scontrino per un ordine su disco fisso (note con limitazioni nel sorgente)
- `static int longestLine(Order order)` - restituisce la lunghezza massima necessaria per stampare lo scontrino
- `static int longestDish(Order order)` - restituisce la lunghezza del nome di piatto più lungo in quest'ordine
- `static int longestSetting()` - restituisce la lunghezza del nome dell'impostazione più lunga
- `static void appendLeftPad(StringBuilder builder, String toAppend, int lineLen)` - appende del testo allineato a destra
- `static void appendCenterPad(StringBuilder builder, String toAppend, int lineLen)` - appende del testo allineato al centro
- `static void appendDishes(StringBuilder builder, Order order, int lineLen)` - appende la lista di piatti allo scontrino
- `static void appendTotal(StringBuilder builder, Order order, int lineLen)` - appende il totale allo scontrino
- `static void appendLine(StringBuilder builder, int lineLen)` - stampa una riga orizzontale

1.3 View

Descrizione delle classi della View dell'applicazione, cioè la parte grafica che permette all'utente di visualizzare o modificare i dati del Model.
Dato il numero ingente di attributi di queste classi, vengono direttamente annotati su uno screenshot della schermata.


it.colella.prestomanager.view
CassaPage
final PriceFormatter formatter
Order currentOrder
Map<Integer, Order> payableOrders
Integer[] payableTables
CassaPage(JFrame)
[...]
CuocoPage
Order currentOrder
[...]
CuocoPage(JFrame)
[...]
CamerierePage
final PriceFormatter formatter
Dish currentDish
Order currentOrder
CamerierePage(JFrame)
[...]
Page
final int GENERIC_SPACING
final int BUTTON_SPACING
JFrame frame
Map<JComponent, Border> borderCache
private boolean showingDebugBorders
Page(JFrame)
Set<JComponent> getComponentsRecursive(JPanel)
void setPage(JFrame, Page)
void switchPage(Page)
final void setShowingDebugBorders(boolean)
JFrame getFrame()
JPanel
DishPage
final int NAME_MAX_LENGTH
final int DESC_MAX_LENGTH
final int MODE_MODIFY
final int MODE_NEW
final int mode
Dish startingDish
Map<JCheckBox, Allergen> checkBoxMap
[...]
DishPage(JFrame)
DishPage(JFrame, Dish)
[...]
ChefPage
final PriceFormatter formatter
[...]
ChefPage(JFrame)
[...]
MenuPage
final int MAIN_BUTTON_SIZE
[...]
MenuPage(JFrame)
[...]

1.3.1 Page

Descrive una pagina, cioè un JPanel che occupa e gestisce tutto lo spazio fornito dal JFrame. estende `JPanel`. Attributi:

- `static final int GENERIC_SPACING` - specifica una spaziatura uniforme tra `JComponent` per tutte le classi che estendono Page
- `static final int BUTTON_SPACING` - specifica una spaziatura uniforme ridotta, quasi sempre tra pulsanti
- `JFrame frame` - frame gestito dalla pagina
- `Map<JComponent, Border> borderCache` - mappa per poter ripristinare i bordi originali di un JComponent e togliere quelli di debug
- `boolean showingDebugBorders` - se la pagina corrente sta disegnando i bordi di debug

Metodi:

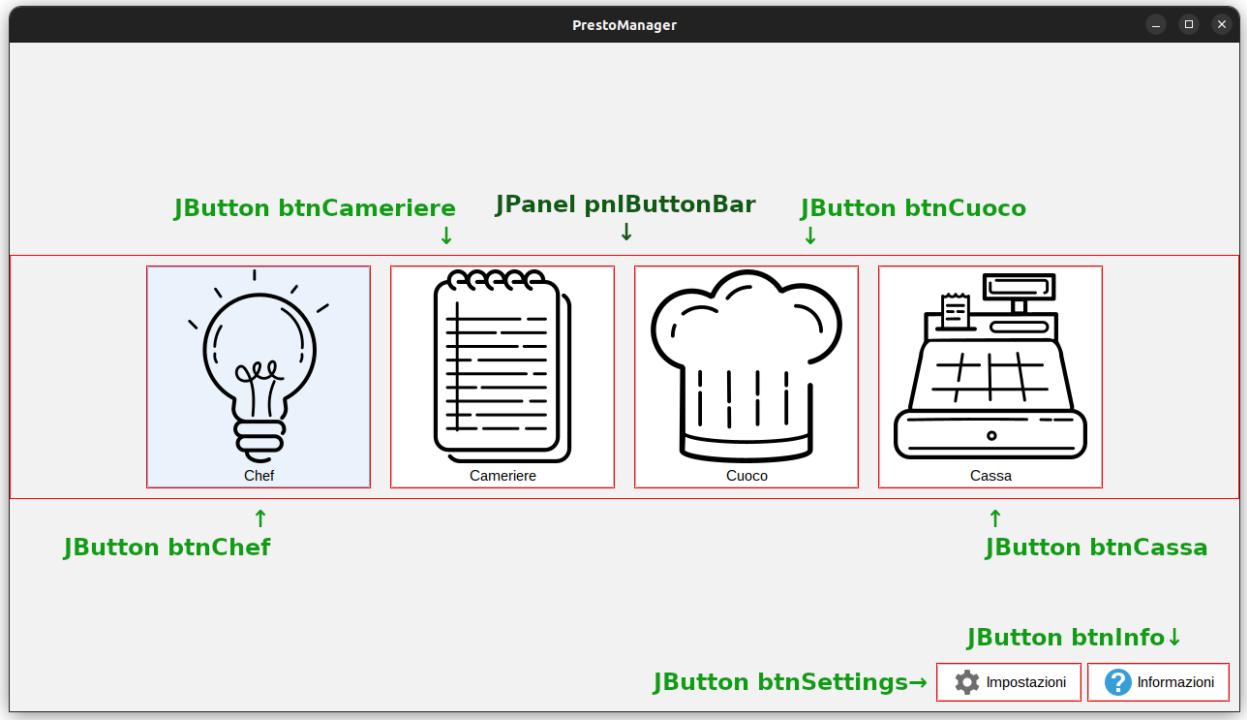
- `static void setPage(JFrame frame, Page page` - Imposta la pagina corrente per un JFrame, utile subito dopo la sua creazione
- `Page(JFrame frame)` - imposta `frame`, inizializza `showingDebugBorders` e `borderCache` a `false` e una mappa vuota, fa in modo che quando l'utente preme **F3** i bordi di debug passino da invisibili a visibili o viceversa
- `final void setShowingDebugBorders(boolean value)` - Imposta se mostare o meno i bordi di debug della pagina
- `static Set<JComponent> getJComponentsRecursive(JPanel c)` - Ottiene tutti i JComponent in un JPanel c. Se uno dei componenti di c è anch'esso un JPanel, ottiene anche tutti i JComponent di quel panel, e così via.

Inoltre tutte le sottoclassi di Page contengono (per convenienza, non imposto da Page):

- `void initActions()` - inizializza le `Action` della pagina. Chiamato il prima possibile nel costruttore, prima della creazione di qualsiasi `JComponent`
- `void initPostActions()` - usato solo da alcune schermate, inizializza delle Action alla fine del costruttore, dopo aver creato tutti i `JComponent`
- `Action actionShortcutInfo` - action definita localmente all'interno di `initActions()`, mostra un riepilogo delle scorciatoie da tastiera per la pagina corrente *("F1")*

1.3.2 MenuPage

Pagina principale. Consente inoltre di modificare le impostazioni e leggere informazioni sul programma e sulle librerie usate



Altri attributi e Action:

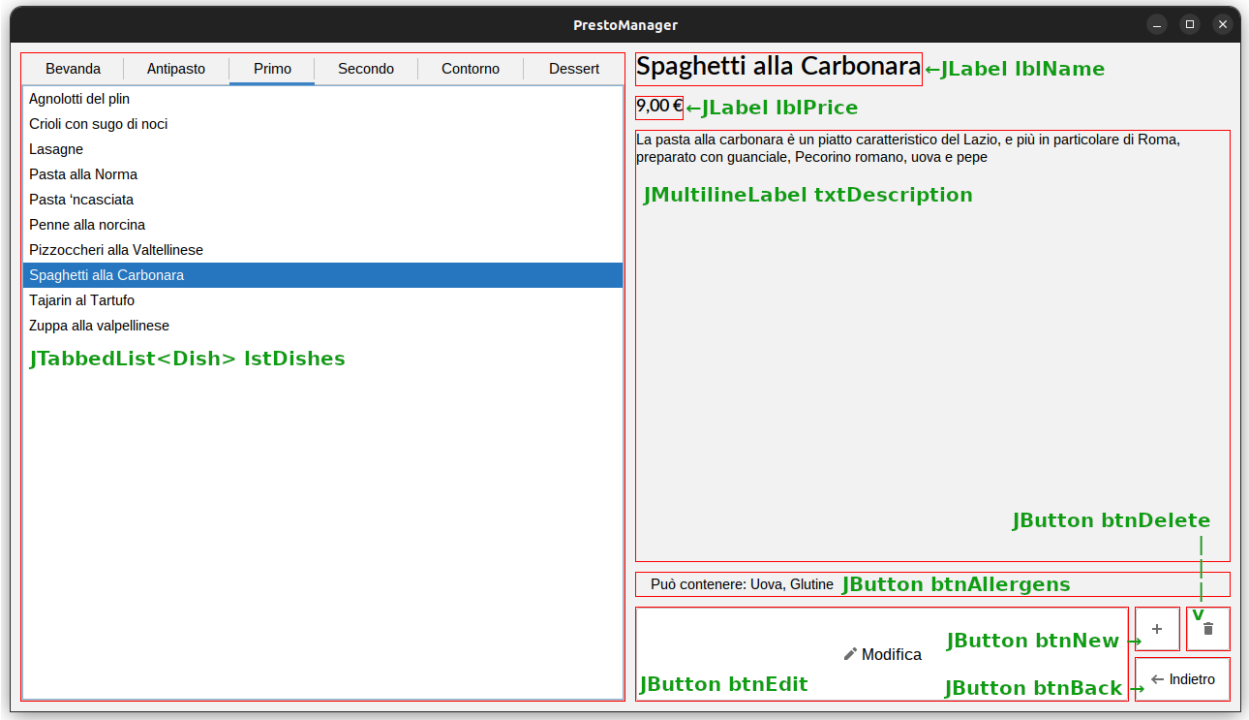
- `static final int MAIN_BUTTON_SIZE` - dimensione dei pulsanti principali
- `Action actionChef` - passa alla schermata ChefPage *(btnChef o "1")*
- `Action actionCameriere` - passa alla schermata CamerierePage *(btnCameriere o "2")*
- `Action actionCuoco` - passa alla schermata CuocoPage *(btnCuoco o "3")*
- `Action actionCassa` - passa alla schermata CassaPage - *(btnCassa o "4")*
- `Action actionSettings` - apre un popup per modificare le impostazioni *(btnSettings o "S")*
- `Action actionInfo` - apre un popup con informazioni sul programma e opzionalmente sulle librerie usate *(btnInfo o "I")*

Metodi:

- `MenuPage(JFrame frame)` - costruttore, se è la prima volta che l'utente apre il programma apre una finestra di benvenuto tramite `showWelcome()`
- `void showWelcome()` - apre una finestra di benvenuto, in seguito apre la finestra di impostazioni tramite `showSettings()`
- `void showSettings()` - apre una finestra per modificare le impostazioni (vedi Setting)

1.3.3 ChefPage

Pagina per la visualizzazione del menù del ristorante. La modifica di piatti è effettuata in DishPage



Altri attributi e Action:

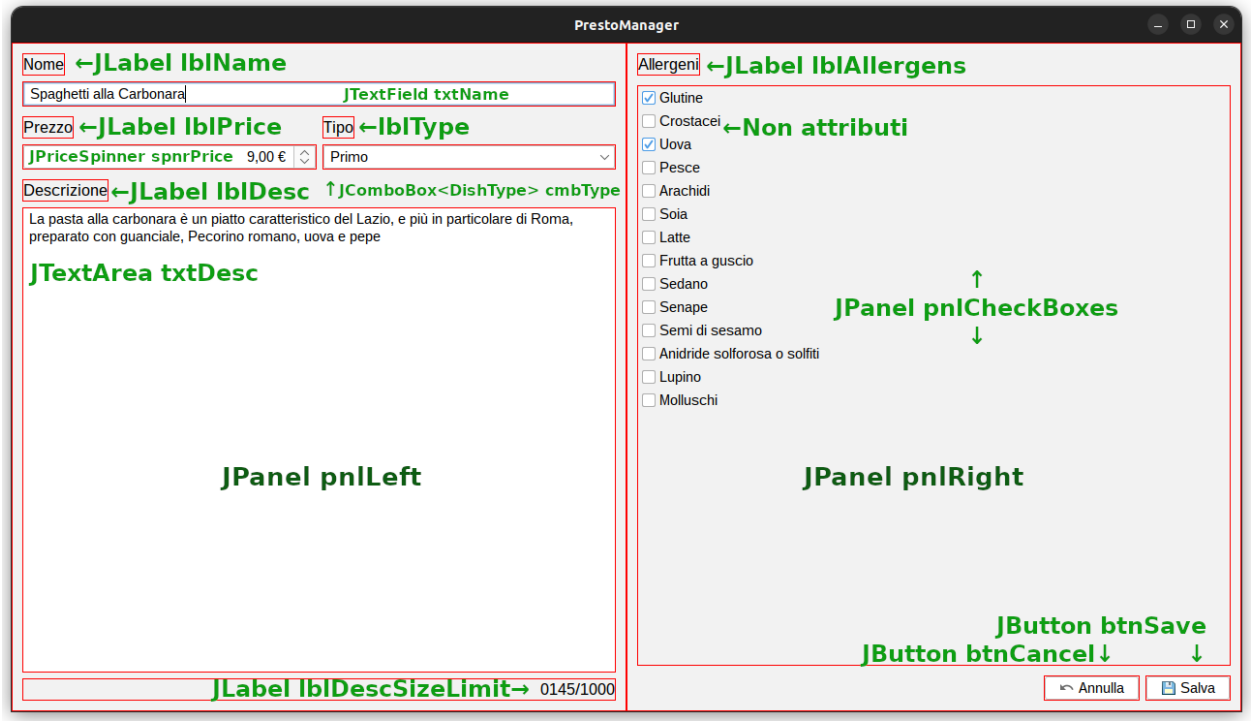
- `static final PriceFormatter formatter` - necessario per formattare il prezzo
- `Action actionShowAllergens` - apre un JAllergenInfoPane per il piatto selezionato *(btnAllergens)*
- `Action actionEdit` - passa alla schermata `DishPage` per il piatto selezionato *(btnEdit o "Enter")*
- `Action actionNew` - passa alla schermata `DishPage` per creare un nuovo piatto *(btnNew o "Ctrl+N")*
- `Action actionDelete` - vedi `onDelete()` *(btnDelete o "Canc")*
- `Action actionBack` - torna al menù principale *(btnBack o "Esc")*

Metodi:

- `void onSelectionChange()` - chiamato quando cambia il piatto selezionato, aggiorna le informazioni mostrate
- `void onDelete()` - apre una finestra di conferma, poi cancella il piatto selezionato
- `void initTabbedList()` - inizializza o reinizializza lstDishes
- `void updateAllergenInfo(Dish d)` - aggiorna il testo mostrato da btnAllergens

1.3.4 DishPage

Pagina per la creazione o modifica di piatti del menù



Altri attributi e Action:

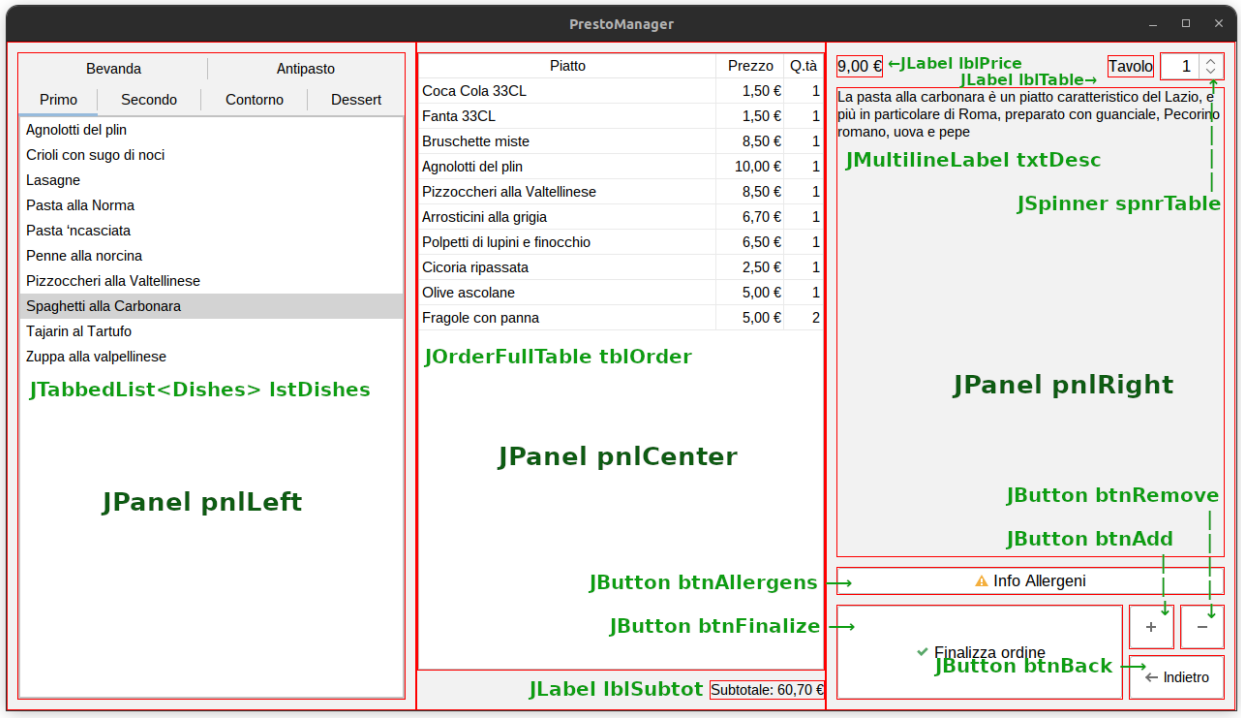
- `static final int NAME_MAX_LENGTH` - lunghezza massima del nome di un piatto
- `static final int DESC_MAX_LENGTH` - lunghezza massima della descrizione di un piatto
- `static final int MODE_MODIFY, MODE_NEW` - costanti, vedi `mode`
- `final int mode` - determina se stiamo creando un piatto nuovo (`MODE_NEW`) o modificandone uno esistente (`MODE_MODIFY`)
- `Dish startingDish` - piatto col quale è stata inizializzata la pagina
- `Map<JCheckBox, Allergen> checkBoxMap` - mappa per la lettura dello stato di ogni checkbox
- `Action actionCancel` - vedi `onCancel()` *(btnCancel o "Esc")*
- `Action actionSave` - vedi `onSave()` *(btnSave o "Ctrl+S")*

Metodi:

- `DishPage(JFrame frame)` - crea una DishPage per la creazione di un nuovo piatto
- `DishPage(JFrame frame, Dish dish)` - crea una DishPage per la modifica di un piatto esistente
- `DishPage(JFrame frame, Dish dish, int mode` - usato internamente
- `void onCancel()` - torna a `ChefPage`. Se il piatto corrente è stato modificato, chiede prima conferma
- `void onSave()` - salva il piatto corrente e, in base alla modalità, torna a `ChefPage`
- `void updateDescLen()` - aggiorna lblDescSizeLimit
- `Dish buildDishFromInput()` - crea un oggetto Dish dai valori inseriti dall'utente.

1.3.5 CamerierePage

Pagina per la creazione di ordini per un tavolo



Altri attributi e Action:

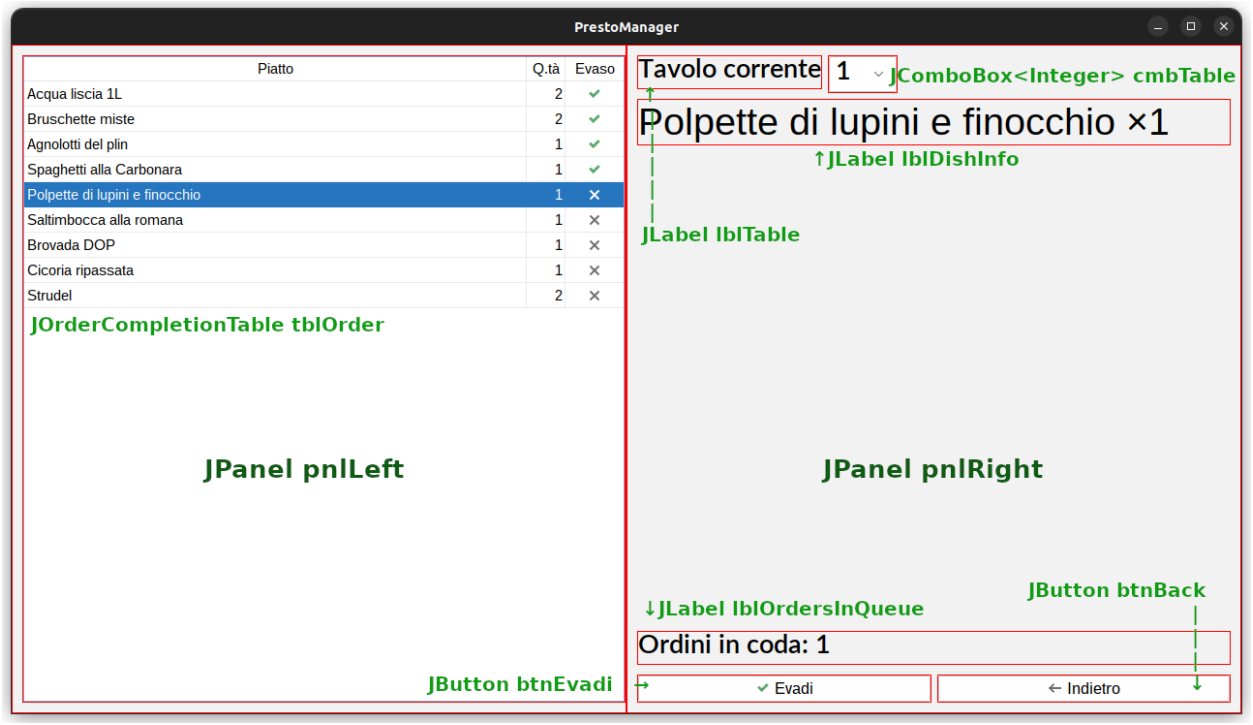
- `static final PriceFormatter formatter` - necessario per formattare il prezzo
- `Dish selectedDish` - il piatto al selezionato dall'utente
- `Order currentOrder` - l'ordine che stiamo creando
- `Action actionShowAllergens` - apre un JAllergenInfoPane con le informazioni sugli allergeni del piatto selezionato *(btnAllergens)*
- `Action actionFinalize` - vedi `onFinalize()` *(btnFinalize o "Shift+Enter")*
- `Action actionAdd` - vedi `onAdd()` *(btnAdd, "Enter" o "+")*
- `Action actionRemove` - vedi `onRemove()` *(btnRemove o "-")*
- `Action actionBack` - vedi `onBack()` *(btnBack o "Esc")*

Metodi:

- `CamerierePage(JFrame frame)` - crea una CamerierePage
- `void onSelectionChange(Dish newSelection)` - aggiorna la GUI quando cambia il piatto selezionato
- `void onFinalize()` - finalizza l'ordine corrente e lo passa al cuoco (previa conferma)
- `void onAdd()` - aggiunge un piatto all'ordine corrente
- `void onRemove()` - rimuove un piatto dall'ordine corrente
- `void onBack()` - torna al menù principale. Se l'ordine corrente non è vuoto, chiede prima conferma
- `void calculateSubTotal()` - Ricalcola il subtotale e lo stampa in `lblSubTot`

1.3.6 CuocoPage

Pagina per l'evasione di ordini da parte del cuoco



Altri attributi e Action:

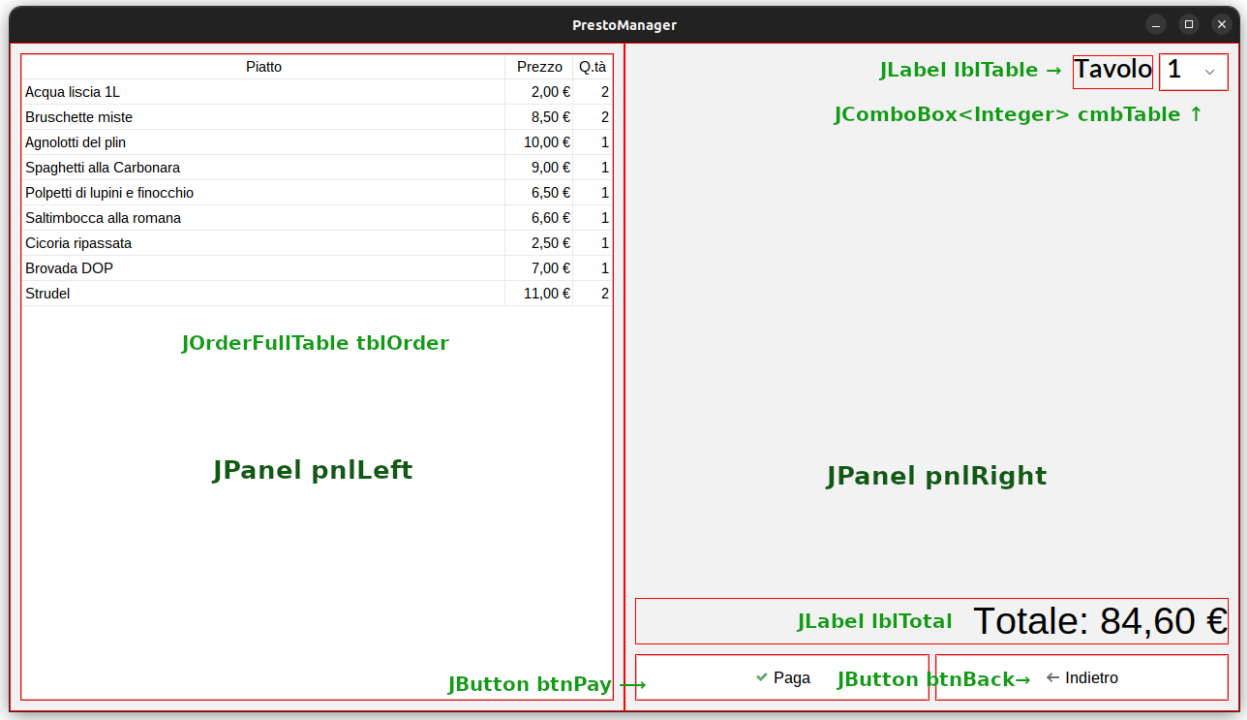
- `Order currentOrder` - ordine corrente visualizzato
- `Action actionEvadi` - vedi `onEvadi()` *(btnEvadi o "Enter")*
- `Action actionBack` - vedi `onBack()` *(btnBack o "Esc")*

Metodi:

- `CuocoPage(JFrame frame)` - crea una CuocoPage
- `void onEvadi()` - segna il piatto selezionato come completato o meno. Se tutti i piatti sono stati evasi, manda l'ordine in cassa e va all'ordine successivo
- `void onBack()` - torna al menù principale. Se almeno un piatto è stato marcato come evaso, chiede prima conferma
- `void onSelectionChange()` - aggiorna la GUI quando il tavolo selezionato cambia

1.3.7 CassaPage

Pagina per pagare il conto di un tavolo e stamparne lo scontrino



Altri attributi e Action:

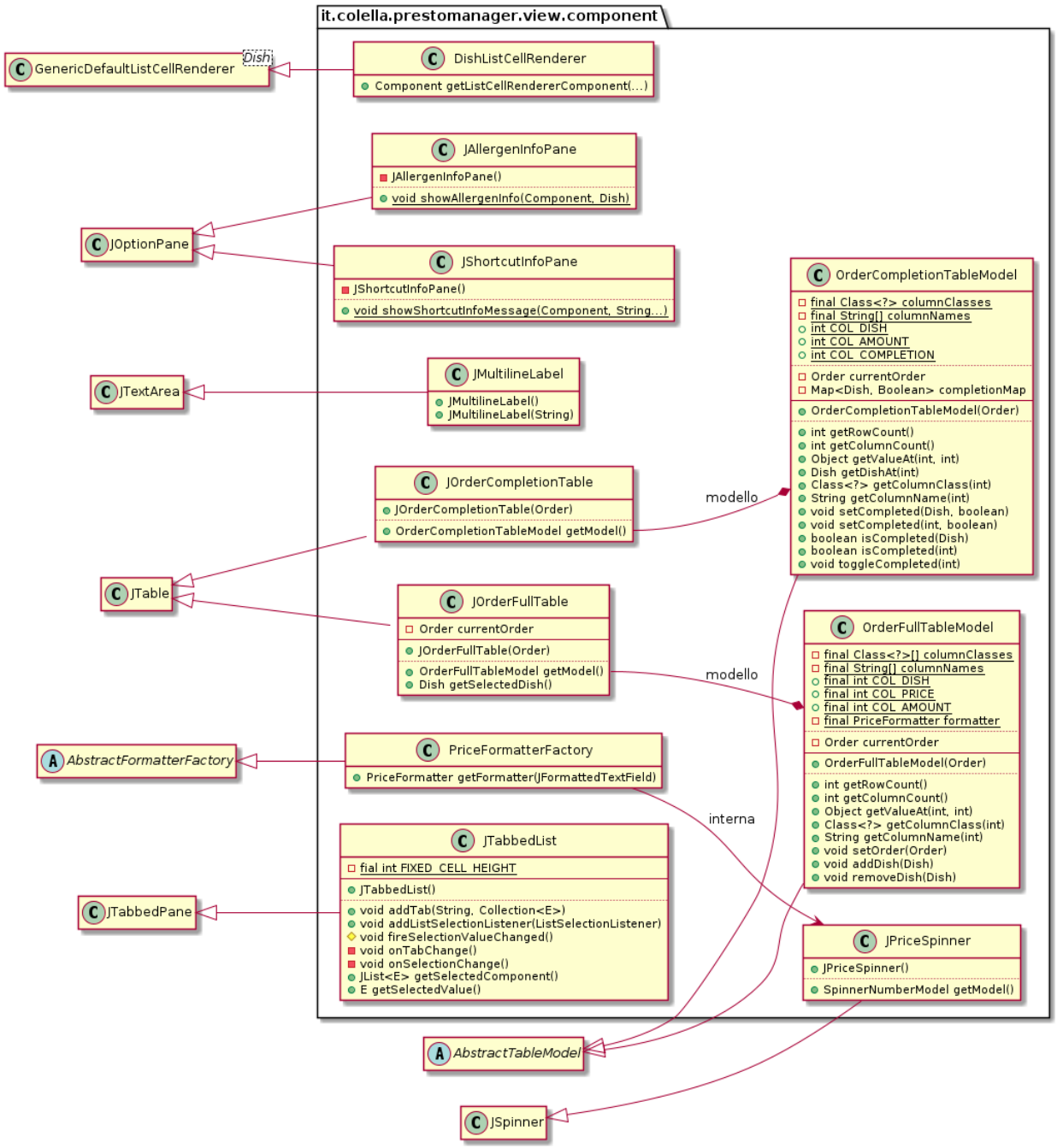
- `static final PriceFormatter formatter` - necessario per formattare il prezzo
- `Order currentOrder` - l'ordine selezionato da pagare
- `Map<Integer, Order>` payableOrders - mappa {numero tavolo : Ordine} degli ordini pagabili. Vedi OrderManager::getPayableOrders
- `Integer[] payableTables` - array dei tavoli con conti pagabili. Vedi OrderManager::getPayableTables
- `Action actionPay` - vedi `onPay()` *(btnPay o "Enter")*
- `Action actionBack` - torna al menù principale

Metodi:

- `CassaPage(JFrame frame)` - crea una CassaPage
- `void nextTable()` e `void prevTable()` - vanno al conto del tavolo successivo/precedente quando l'utente preme la freccetta destra/sinistra
- `void onTableChange()` - aggiorna la GUI quando cambia l'ordine selezionato
- `void onPay()` - paga il conto del tavolo selezionato e ne stampa lo scontrino, poi passa al tavolo successivo
- `void calculateTotal()` - calcola il totale e lo stampa in `lblTotal`

1.4 Component

Le classi Component sono i blocchi di base usate per formare la GUI



1.4.1 DishListCellRenderer

ListCellRenderer per oggetti Dish. Ha solo un metodo:

- `@Override Component getListCellRendererComponent(...)` - restituisce un JLabel con testo del nome del piatto

1.4.2 JAllergenInfoPane

JOptionPane contenente una lista di allergeni del piatto selezionato. Costruttore privato vuoto, ha solo un metodo:

- `static void showAllergenInfo(Component parentComponent, Dish dish)`

1.4.3 JShortcutInfoPane

JOptionPane contenente una lista di scorciatoia da tastiera per la pagina corrente. Costruttore privato vuoto, ha solo un metodo:

- `static void showShortcutInfoMessage(Component parentComponent, String... message)` - usato ad esempio come

```
JShortcutInfoPane.showShortcutInfoMessage(this,
"1", "Chef",
"2", "Cameriere",
"3", "Cuoco",
"4", "Cassa");
```

1.4.4 JMultilineLabel

JTextArea con proprietà preimpostate per apparire come un JLabel multilinea. Non ha attributi. Metodi:

- `JMultilineLabel()` - crea un JMultilineLabel senza testo
- `JMultilineLabel(String text)` - crea un JMultilineLabel partendo dal testo specificato

1.4.5 JOrderCompletionTable

JTable che tramite un OrderCompletionTableModel raffigura un ordine e lo stato di completezza di ogni suo piatto. Non ha attributi. Metodi:

- `JOrderCompletionTable(Order order)` - crea una tabella dall'ordine specificato con nessun piatto evaso
- `@Override OrderCompletionTableModel getModel()` - sovrascritto per restituire il suo OrderCompletionTableModel anziché un TableModel generico

1.4.6 OrderCompletionTableModel

TableModel per rappresentare tutti i piatti di un ordine con il loro stato di completezza. Attributi:

- `static final Class<?>[] columnClasses` - classi degli oggetti in ogni colonna
- `static final String[] columnNames` - nomi delle colonne
- `static final COL_DISH, COL_AMOUNT, COL_COMPLETION` - costanti per semplificare l'uso dei metodi che richiedono il numero della colonna
- `Order currentOrder` - l'ordine a cui si riferisce il modello
- `Map<Dish, Boolean> completionMap` - per ogni piatto segna se è stato evaso o no

Metodi:

- `OrderCompletionTableModel(Order order)` - inizializza `currentOrder` e `completionMap`
- `@Override getColumnCount()` - restituisce il numero di colonne della tabella
- `@Override getRowCount()` - restituisce il numero di righe della tabella
- `@Override Object getValueAt(int rowIndex, int columnIndex)` - restituisce l'oggetto individuato nella tabella dalla riga `rowIndex` e dalla colonna `columnIndex`
- `Dish getDishAt(int rowIndex)` - restituisce il piatto raffigurato in una riga

- `@Override Class<?> getColumnClass(int c)` - restituisce il tipo di oggetti della colonna `c`
- `@Override String getColumnName(int column)` - restituisce il nome della colonna `column`
- `void setCompleted(Dish d, boolean value)` e `void setCompleted(int row, boolean value` - segnano il piatto d (oppure il piatto alla riga row) come completato o meno per l'ordine corrente
- `boolean isCompleted(Dish d)` e `boolean isCompleted(int row)` - restituiscono se il piatto d (oppure il piatto alla riga row) è evaso o meno
- `void toggleCompleted(int row)` - segna il piatto alla riga `row` come evaso se non lo era, o viceversa
- `void setOrder(Order newOrder)` - imposta l'ordine raffigurato dal modello e ricostruisce la tabella. Le informazioni sulla completezza dei piatti sono scartate

1.4.7 JOrderFullTable

JTable che tramite un OrderFullTableModel raffigura un ordine. Attributi:

- `Order currentOrder` - l'ordine raffigurato

Metodi:

- `JOrderFullTable` - crea una tabella dall'ordine specificato
- `@Override OrderFullTableModel getModel()` - sovrascritto per restituire il suo OrderFullTableModel anziché un TableModel generico
- `Dish getSelectedDish()` - restituisce il piatto selezionato

1.4.8 OrderFullTableModel

TableModel per rappresentare tutti i piatti di un ordine. Attributi:

- `static final Class<?>[] columnClasses` - classi degli oggetti in ogni colonna
- `static final String[] columnNames` - nomi delle colonne
- `static final COL_DISH, COL_PRICE, COL_AMOUNT` - costanti per semplificare l'uso dei metodi che richiedono il numero della colonna
- `static final PriceFormatter formatter` - necessario per formattare il prezzo
- `Order currentOrder` - l'ordine a cui si riferisce il modello

Metodi:

- `OrderFullTableModel(Order order)` - crea un OrderFullTableModel dall'ordine specificato
- `@Override getColumnCount()` - restituisce il numero di colonne della tabella
- `@Override getRowCount()` - restituisce il numero di righe della tabella
- `@Override Object getValueAt(int rowIndex, int columnIndex)` - restituisce l'oggetto individuato nella tabella dalla riga `rowIndex` e dalla colonna `columnIndex`
- `@Override Class<?> getColumnClass(int c)` - restituisce il tipo di oggetti della colonna `c`
- `@Override String getColumnName(int column)` - restituisce il nome della colonna `column`
- `void setOrder(Order newOrder)` - imposta l'ordine raffigurato dal modello e ricostruisce la tabella
- `void addDish(Dish d)` - vedi `Order::add(Dish d)`
- `void removeDish(Dish d)` - vedi `Order::remove(Dish d)`

1.4.9 JPriceSpinner

JSpinner specializzato nella visualizzazione e input di valori di prezzo (double). Consente di selezionare tutto il contenuto dello spinner con un solo click e di modificare il valore con la rotella del mouse. Non ha attributi. Metodi:

- `JPriceSpinner()` - Crea un JPriceSpinner con valore limitato tra 0 e 999.99 a intervalli di 0.5
- `@Override getModel()` - sovrascritto per restituire il suo SpinnerNumberModel anziché uno `SpinnerModel` generico

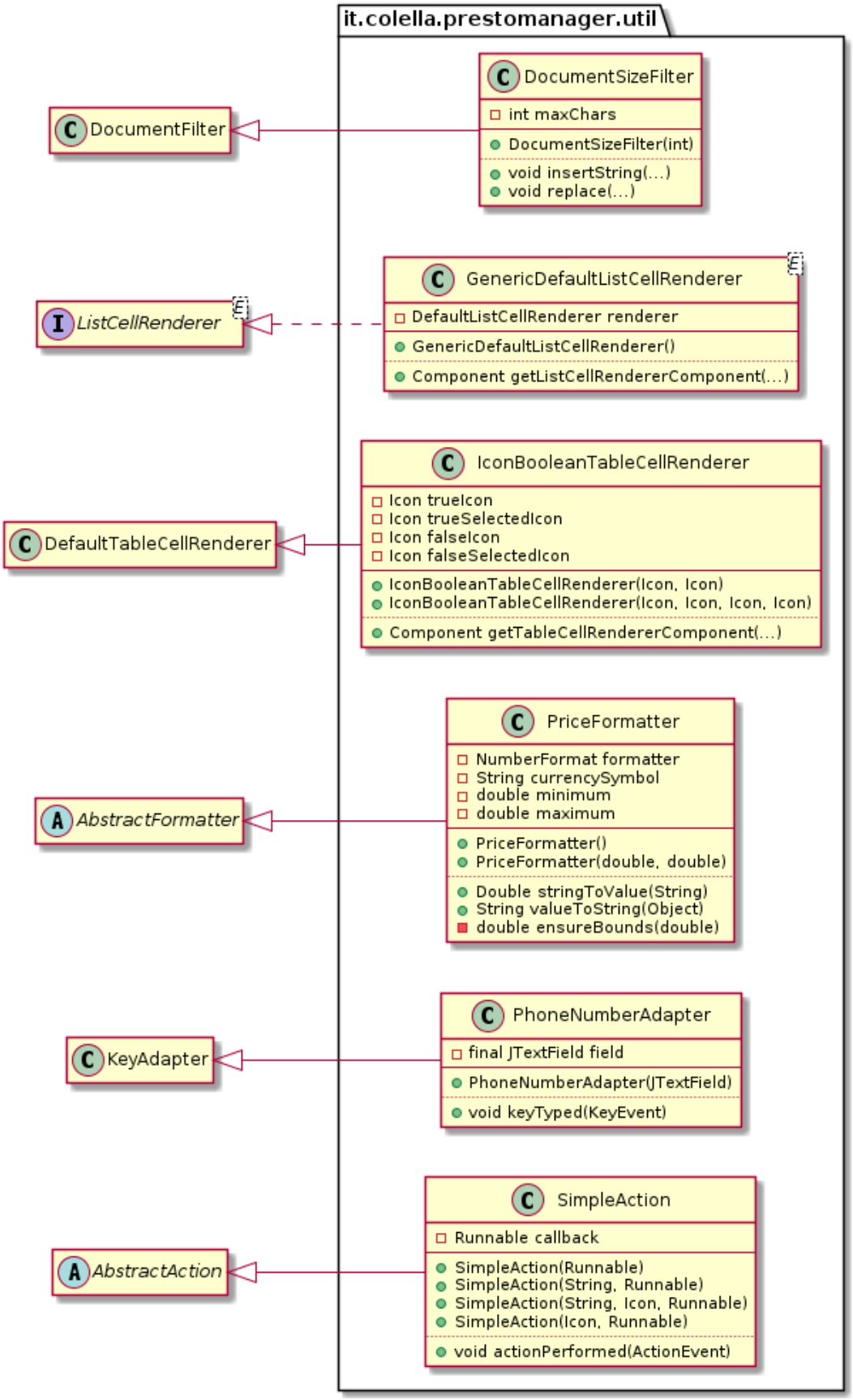
1.4.10 PriceFormatterFactory

Non si può impostare direttamente un formatter all'editor di un JSpinner, bisogna usare una sottoclasse di AbstractFormatterFactory. Ha solo un metodo:

- `@Override PriceFormatter getFormatter(JFormattedTextField tf)` - restituisce il formatter già presente nel text field se è un PriceFormatter, altrimenti ne crea uno nuovo

1.5 Util

Classi non specifiche a questo progetto, facilmente riutilizzabili in futuro


it.colella.prestomanager.util
DocumentSizeFilter
int maxChars
DocumentSizeFilter(int)
void insertString(...)
void replace(...)
DocumentFilter
GenericDefaultListCellRenderer
E
DefaultListCellRenderer renderer
GenericDefaultListCellRenderer()
Component getListCellRendererComponent(...)
ListCellRenderer
E
IconBooleanTableCellRenderer
Icon trueIcon
Icon trueSelectedIcon
Icon falseIcon
Icon falseSelectedIcon
IconBooleanTableCellRenderer(Icon, Icon)
IconBooleanTableCellRenderer(Icon, Icon, Icon, Icon)
Component getTableCellRendererComponent(...)
DefaultTableCellRenderer
PriceFormatter
NumberFormat formatter
String currencySymbol
double minimum
double maximum
PriceFormatter()
PriceFormatter(double, double)
Double stringToValue(String)
String valueToString(Object)
double ensureBounds(double)
AbstractFormatter
PhoneNumberAdapter
final JTextField field
PhoneNumberAdapter(JTextField)
void keyTyped(KeyEvent)
KeyAdapter
SimpleAction
Runnable callback
SimpleAction(Runnable)
SimpleAction(String, Runnable)
SimpleAction(String, Icon, Runnable)
SimpleAction(Icon, Runnable)
void actionPerformed(ActionEvent)
AbstractAction

1.5.1 DocumentSizeFilter

DocumentFilter che limita il numero massimo di caratteri consentiti in un documento. Attributi:

- `int maxChars` - numero massimo di caratteri consentiti

Metodi:

- `DocumentSizeFilter(int maxChars)` - imposta maxChars
- `@Override void insertString(...)` - blocca l'inserimento di nuovi caratteri se superano la lunghezza massima
- `@Override void replace(...)` - quando viene incollato del testo, inserisce caratteri finché abbiamo spazio e poi tronca

1.5.2 GenericDefaultListCellRenderer

Permette di usare un DefaultListCellRenderer per JList con tipo generico. Attributi:

- `<E>` - il tipo della JList alla quale si assegna questo renderer
- `DefaultListCellRenderer` - renderer usato internamente

Metodi:

- `GenericDefaultListCellRenderer()` - imposta `renderer`
- `@Override Component getListCellRendererComponent(...)` - delega a `renderer`

1.5.3 IconBooleanTableCellRenderer

extends DefaultTableCellRenderer Renderer per oggetti `Boolean` per celle di una JTable. Supporta opzionalmente icone differenti per quando la cella è selezionata. Attributi:

- `Icon trueIcon` - icona da usare per `true` se la cella non è selezionata
- `Icon trueSelectedIcon` - icona da usare per `true` se la cella è selezionata
- `Icon falseIcon` - icona da usare per `false` se la cella non è selezionata
- `Icon falseSelectedIcon` - icona da usare per `false` se la cella è selezionata

Metodi:

- `IconBooleanTableCellRenderer(Icon trueIcon, Icon falseIcon, Icon trueSelectedIcon, Icon falseSelected` - imposta le icone. `trueSelectedIcon` e `falseSelectedIcon` sono parametri opzionali. In quel caso internamente `trueIcon` e `trueSelectedIcon` saranno uguali, come `falseIcon` e `falseSelectedIcon`
- `@Override Component getTableCellRendererComponent(...)` - restituisce il JLabel con l'icona appropriata da stampare nella tabella

1.5.4 PhoneNumberAdapter

KeyAdapter che consente solo numeri di telefono nel `JTextField` collegato. Permette di inserire un carattere se e solo se è uno dei seguenti:

- Un + all'inizio
- Uno spazio non all'inizio e non di seguito ad un altro spazio
- Una cifra

Attributi:

- `JTextField field` - JTextField collegato

Metodi:

- `PhoneNumberAdapter(JTextField field)` - imposta `field`
- `@Override void keyTyped(KeyEvent e)` - blocca il KeyEvent se il carattere non è consentito

1.5.5 PriceFormatter

Formatter per convertire quantità di denaro da String a Double e viceversa. Attributi:

- `NumberFormat formatter` - formatter usato internamente per convertire da Double a String
- `String currencySymbol` - simbolo della moneta usata ('€', '$', '£', etc.)
- `double minimum` - valore minimo consentito
- `double maximum` - valore massimo consentito

Metodi:

- `PriceFormatter()` - imposta `currencySymbol`. Imposta `minimum` e `maximum` a `Double.MIN_VALUE` e `Double.MAX_VALUE`
- `PriceFormatter(double minimum, double maximum)` - imposta tutti gli attributi
- `@Override Double stringToValue(String text)` - converte da String a Double, consentendo l'uso intercambiabile della virgola e del punto come delimitatori decimali. Output limitato tra minimo e massimo
- `@Override String valueToString(Object value)` - delega a `NumberFormat::format`, aspettandosi un input di tipo Double
- `double ensureBounds(double value)` - usato per garantire che l'output sia tra minimo e massimo

1.5.6 SimpleAction

Semplifica la creazione di Action consentendo di usare lambda o method reference

```
Action action = new SimpleAction(this::performAction);
// Oppure
Action action = new SimpleAction((arg) -> this.performAction(arg));
// Invece di
Action action = new AbstractAction() {
        @Override
        public void actionPerformed(ActionEvent e) {
                this.performAction();
        }
};
```

Attributi:

- `Runnable callback` - funzione chiamata su `actionPerformed()`

Metodi:

- Vari costruttori - impostano `callback`
- `@Override void actionPerformed(ActionEvent e)` - esegue `callback`

2 Test

Classi di test per verificare il corretto funzionamento del programma

2.1 DishTest

Test per Dish:

- Un test che verifica che nomi più lunghi di quanto consentito da GUI funzionino comunque (nel caso venga modificato direttamente il JSON)
- Un test che verifica che descrizioni più lunghe di quanto consentito da GUI funzionino comunque (nel caso venga modificato direttamente il JSON)
- Un test che verifica che prezzi negativi funzionino comunque (Un test che verifica che nomi più lunghi di quanto consentito da GUI funzionino comunque (nel caso venga modificato direttamente il JSON)
- Due test per il corretto funzionamento di `equals()`
- Sei test per il corretto funzionamento di `deepEquals()`
- Un test per verificare che il set restituito da `getAllergens()` non sia modificabile

2.2 OrderTest

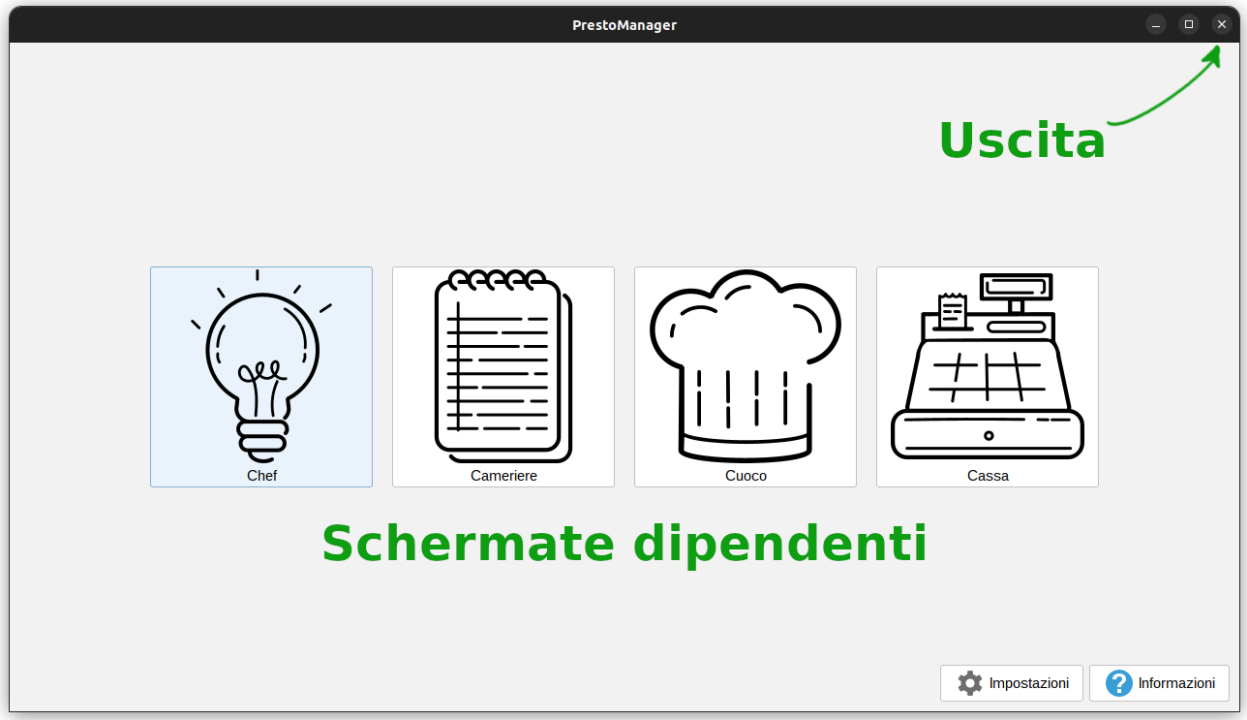
Test per Order:

- Un test per verificare che numeri di tavolo negativi non siano consentiti
- Due test per verificare che aggiungere o rimuovere quantità negative di porzioni di un piatto non sia consentito
- Un test per verificare che la mappa restituita da `getMap()` non sia modificabile

3 Descrizione delle funzionalità

3.1 Menù Principale

Visualizzare un menù iniziale, mostrato all'avvio del programma, che permetta sia l'accesso alle schermate relative ai diversi dipendenti che l'uscita dall'applicazione

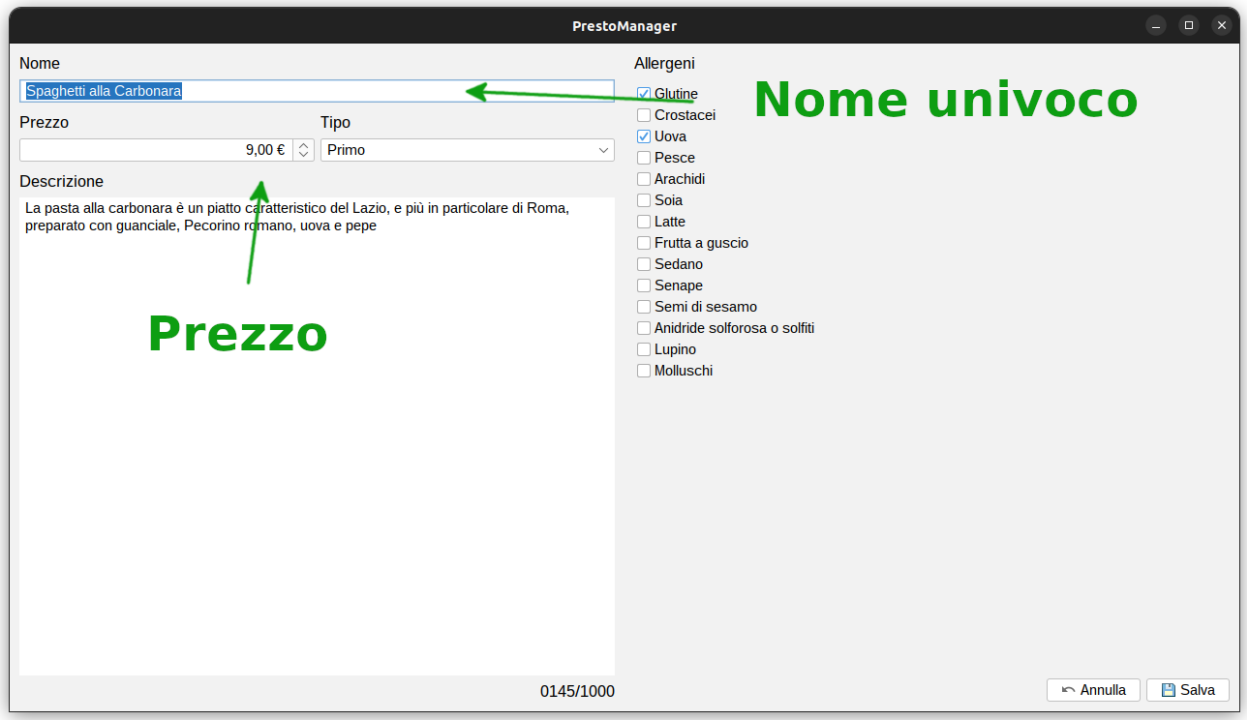


Vista la struttura a "schermate singole" dell'applicazione, inizialmente ho provato ad usare un Layout Manager come CardLayout. Questo ha una conseguenza importante: **è creata una singola istanza persistente per ogni pagina**. Il che è ottimo per la performance, ma vuole anche dire che i cambiamenti di stato dell'applicazione sono molto più difficili da gestire (ad esempio, se impostiamo lo sfondo di ChefPage a rosso e poi torniamo al menù principale, con CardLayout una volta tornati a ChefPage lo sfondo sarà ancora rosso). Per questo alla fine ho deciso di sviluppare Page per gestire i cambiamenti di schermata.

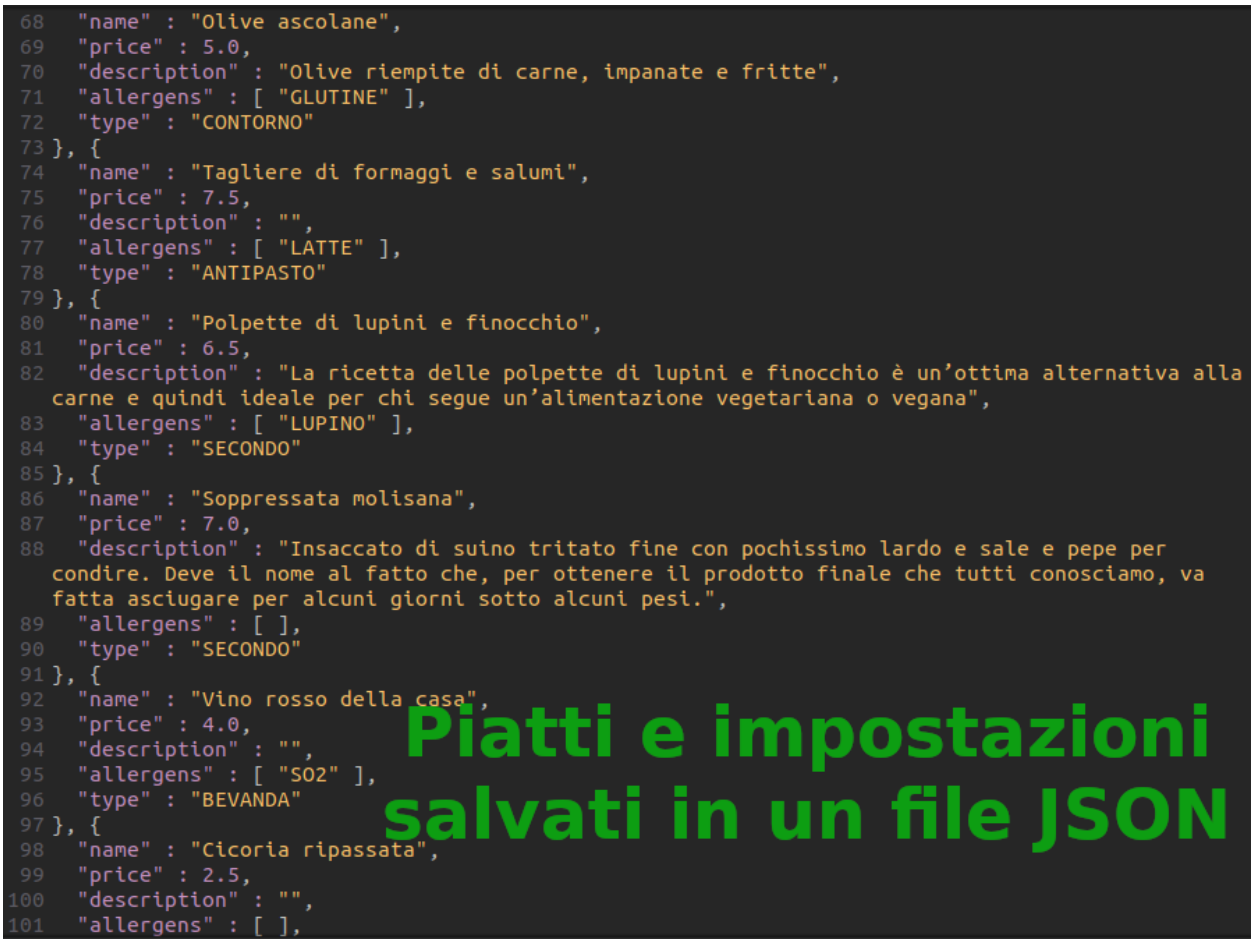
3.2 Chef

Offrire la possibilità di aggiungere, modificare e rimuovere le pietanze all'interno di un menù, identificate da un nome, univoco, ed un prezzo. La lista delle pietanze all'interno di un menù dovrà essere persistita all'interno di un file, in modo tale che le successive esecuzioni del programma non debbano ricostruire nuovamente il menù


PrestoManager
Bevanda
Antipasto
Primo
Secondo
Contorno
Dessert
Agnolotti del plin
Crioli con sugo di noci
Lasagne
Pasta alla Norma
Pasta 'ncasciata
Penne alla norcina
Pizzoccheri alla Valtellinese
Spaghetti alla Carbonara
Tajarin al Tartufo
Zuppa alla valpellinese
Spaghetti alla Carbonara
9,00 €
La pasta alla carbonara è un piatto caratteristico del Lazio, e più in particolare di Roma, preparato con guanciale, Pecorino romano, uova e pepe
Rimuovere
Aggiungere
Può contenere: Glutine, Uova
Modificare
Modifica
+
Indietro


PrestoManager
Nome
Spaghetti alla Carbonara
Prezzo
9,00 €
Tipo
Primo
Descrizione
La pasta alla carbonara è un piatto caratteristico del Lazio, e più in particolare di Roma, preparato con guanciale, Pecorino romano, uova e pepe
Prezzo
0145/1000
Allergeni
Nome univoco
Glutine
Crostacei
Uova
Pesce
Arachidi
Soia
Latte
Frutta a guscio
Sedano
Senape
Semi di sesamo
Anidride solforosa o solfiti
Lupino
Molluschi
Annulla
Salva


PrestoManager
Nome
Spaghetti alla Carbonara
Prezzo
99,00 €
Tipo
Dessert
Descrizione
Piatto con il nome uguale ad uno che già esiste
Allergeni
Glutine
Crostacei
Uova
Pesce
Arachidi
Soia
Latte
Frutta a guscio
Sedano
Senape
Semi di sesamo
Esiste già un piatto con quel nome
OK
Nome duplicato non consentito
0047/1000
Annulla
Salva


"name" : "Olive ascolane",
"price" : 5.0,
"description" : "Olive riempite di carne, impanate e fritte",
"allergens" : ["GLUTINE"],
"type" : "CONTORNO"
}, {
"name" : "Tagliere di formaggi e salumi",
"price" : 7.5,
"description" : "",
"allergens" : ["LATTE"],
"type" : "ANTIPASTO"
}, {
"name" : "Polpette di lupini e finocchio",
"price" : 6.5,
"description" : "La ricetta delle polpette di lupini e finocchio è un'ottima alternativa alla carne e quindi ideale per chi segue un'alimentazione vegetariana o vegana",
"allergens" : ["LUPINO"],
"type" : "SECONDO"
}, {
"name" : "Soppressata molisana",
"price" : 7.0,
"description" : "Insaccato di suino tritato fine con pochissimo lardo e sale e pepe per condire. Deve il nome al fatto che, per ottenere il prodotto finale che tutti conosciamo, va fatta asciugare per alcuni giorni sotto alcuni pesi.",
"allergens" : [ ],
"type" : "SECONDO"
}, {
"name" : "Vino rosso della casa",
"price" : 4.0,
"description" : "",
"allergens" : ["SO2"],
"type" : "BEVANDA"
}, {
"name" : "Cicoria ripassata",
"price" : 2.5,
"description" : "",
"allergens" : [ ],
Piatti e impostazioni salvati in un file JSON

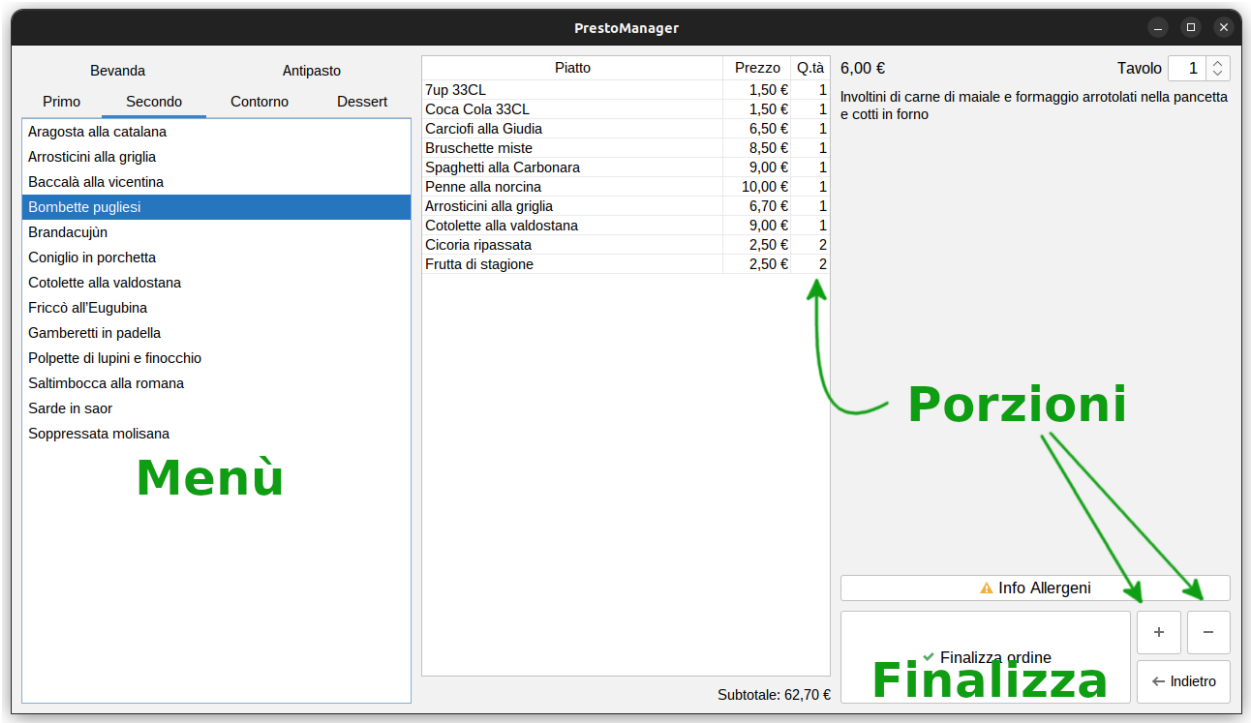
```json
{
  "NAME" : "Trattoria da Zi' Mario",
  "VAT" : "IT12345678901",
  "ADDRESS" : "via del Fosso di Centocelle 50",
  "PHONE" : "+39 12 34 56 789"
}
```

Per far sì che i piatti abbiano un nome univoco ho sfruttato la caratteristica fondamentale dei Set di non ammettere duplicati. Nello specifco, un oggetto o non viene aggiunto al set se $\exists$ *un altro oggetto* o' *nel set tale che* $o.equals(o')$. Per questo ho sovrascritto `equals()` e `hashCode()` in `Dish` in modo tale che due piatti siano uguali (e quini generino lo stesso hash code) se i loro nomi, tolti gli spazi all'inizio e alla fine, e normalizzata la capitalizzazione, sono uguali.

La gestione di I/O di piatti e impostazioni è gestita da DishManager e SettingsManager, che internamente usano Jackson. Lo "spostamento" di ordini da una postazione all'altra è invece gestito da OrderManager

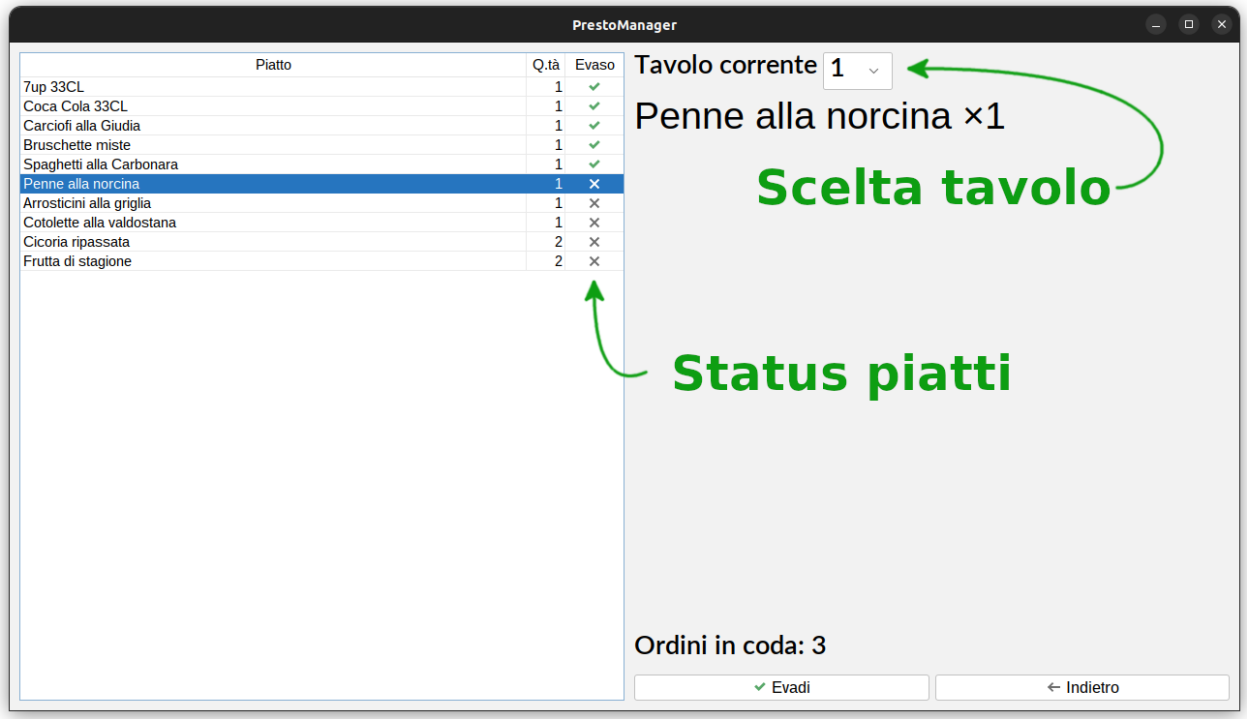
3.3 Cameriere

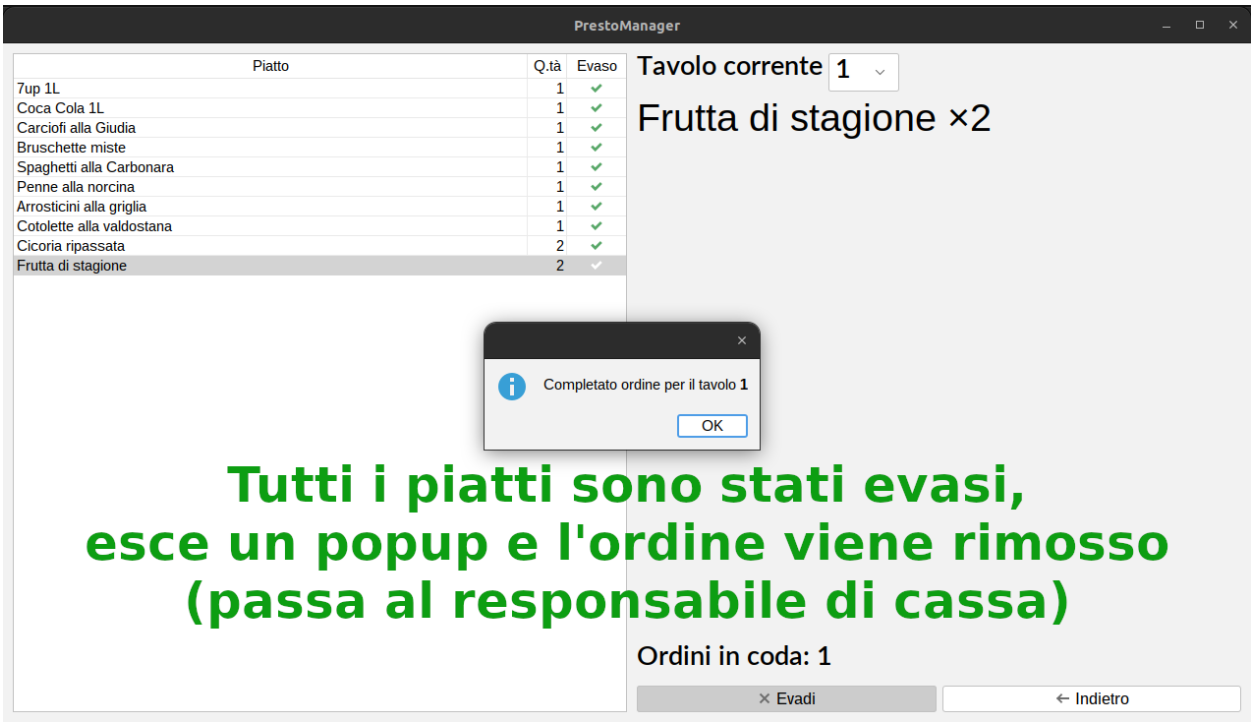
Gestire la creazione di un ordine attraverso una schermata in cui venga visualizzato il menù e nel quale, cliccando su una pietanza, venga aggiunta all'interno dell'ordine con la quantità desiderata. Poiché un cliente può cambiare idea durante l'ordinazione, la creazione di una comanda dovrà prevedere la modifica delle quantità e l'eventuale rimozione di un piatto. Ad ordine finalizzato non sarà più possibile effettuare una modifica



3.4 Cuoco

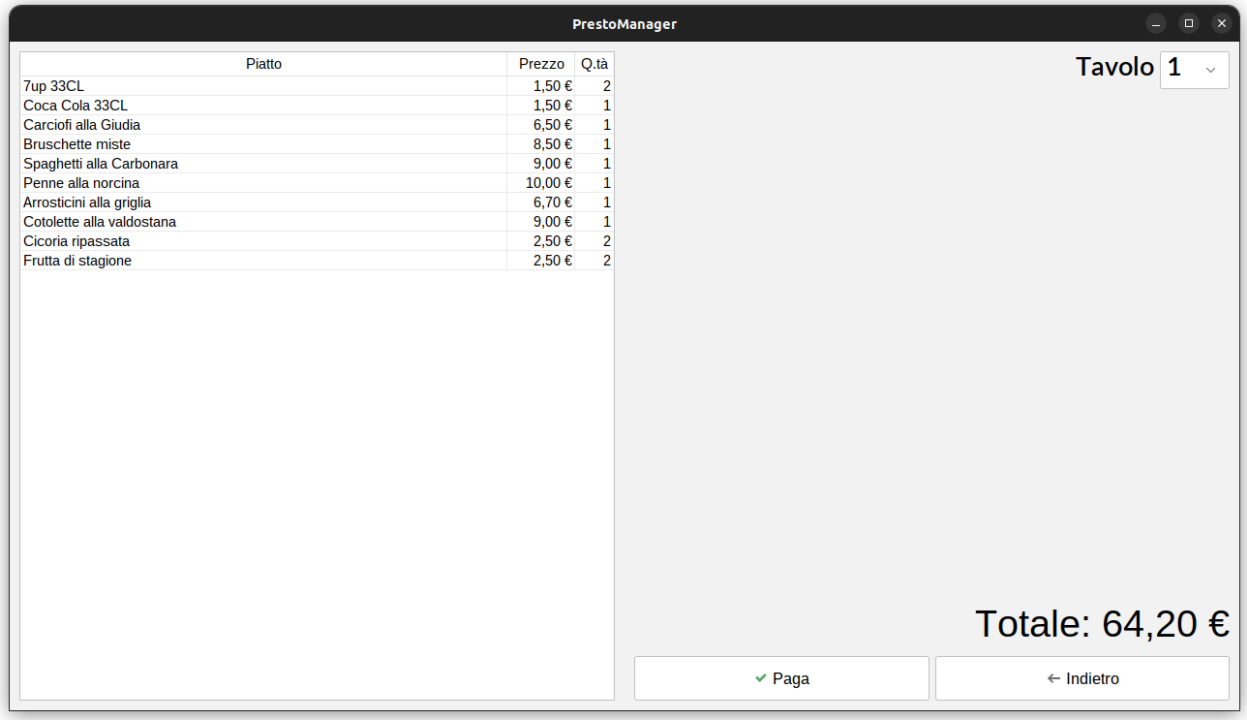
Fornire una schermata per la cucina in cui vengano visualizzati tutti gli ordini di ciascun tavolo, rimuovendo un ordine una volta che tutti i piatti al suo interno sono stati evasi

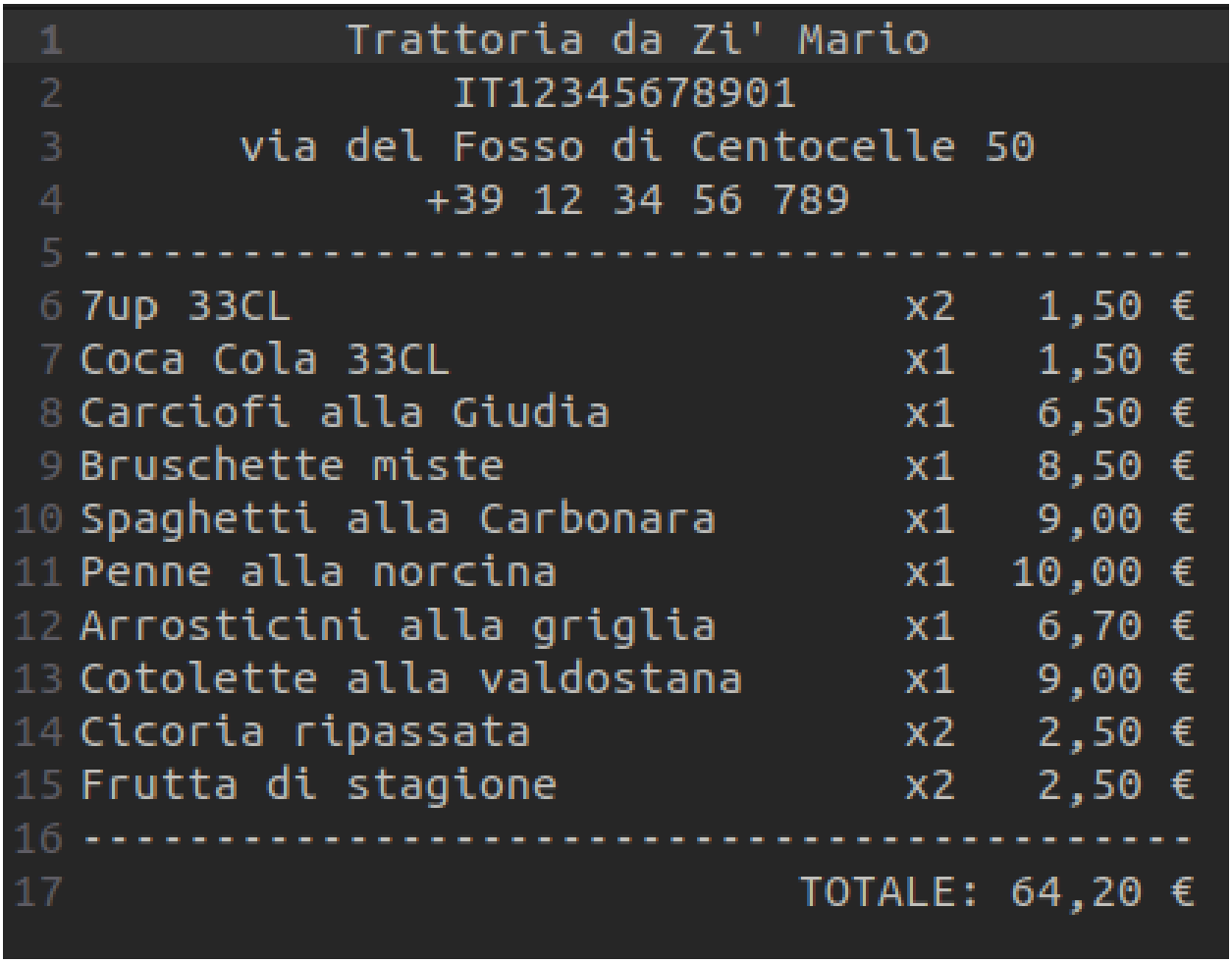




3.5 Responsabile di cassa

Fornire una schermata che permetta il pagamento del conto, selezionandolo da una lista di conti di tutti i tavoli che non hanno ancora effettuato il pagamento. In aggiunta a ciò, dovrà essere simulata la creazione di uno scontrino sotto forma di file di testo. A pagamento effettuato, l'ordine di quel tavolo non dovrà più essere visibile.





Quando un ordine viene evaso (passa quindi dal cuoco alla cassa), OrderManager controlla prima se non ci siano altri ordini già evasi ma non pagati per quel tavolo. Se ce ne sono, unisce i due ordini tramite Order::addAll(), altrimenti segna l'ordine appena evaso come da pagare (lo mette in `cassaOrders`).
Per poter pagare un tavolo deve (riassunto in OrderManager::getPayableTables() e `OrderManager::getPayableOrders()`):

- Avere fatto minimo un ordine (cioè essere in `cassaOrders`)
- Tutti gli ordini di quel tavolo devono essere stati evasi dal cuoco (non si può pagare prima che si abbia ricevuto tutto quello che è stato ordinato)

Se un tavolo non rispetta queste condizioni, non è incluso nella lista dei tavoli che il cassiere può scegliere per pagare il conto