Relazione Progetto

# “Restaurant Manager”

Corso: Programmazione ad oggetti

Anno scolastico 2014/2015

**Autori:** Daniele Rosetti

Sara Sintoni

Matteo Venditto

# Indice

# Analisi

## 1.1 Requisiti

Lo scopo del progetto è quello di realizzare un software che consenta la gestione in modo semplice di un'attività di ristorazione.

Le fasi che gestisce sono:

- Inserimento di una nuova prenotazione inserendo tutti i dati relativi ad essa, verificando la disponibilità del ristorante.
- Rimozione di una prenotazione.
- Aggiunta di un ordine relativo a una prenotazione, scegliendo tra diversi piatti che fanno parte del menù.
- Modifica di un ordine prima del pagamento di esso.
- Pagamento di un ordine in vari metodi.
- Modifica dell'orario in cui si svolge l'esercizio dell'attività di impresa
- Modifica del menù a cui afferiscono le pietanze
- Modifica del numero di posti totali
- Esistenza di un calendario di giornate nel quale non viene svolta l'attività
- I dati devono essere persistenti. Cioè al momento della chiusura del progetto tutti i dati vengono salvati su un file. Sia quelli relativi alle prenotazioni, alle impostazioni del ristorante quali orario… che agli ordini ed anche al menù.

## 1.2 Problema

La principale difficoltà nella creazione del software è la gestione combinata delle varie parti. L'ID della prenotazione deve essere lo stesso dell'ordine, e il pagamento deve conoscere i dati di quest'ultima. Inoltre dopo il pagamento deve essere possibile rimuovere sia l'ordine che la prenotazione. Tutto ciò deve avvenire senza creare eccessive dipendenze tra le varie parti del sistema.

La principale problematica che non affronta il nostro progetto è la sincronizzazione delle varie parti. Nel momento in cui viene aggiunta una prenotazione non è possibile aggiungere un ordine o pagare.

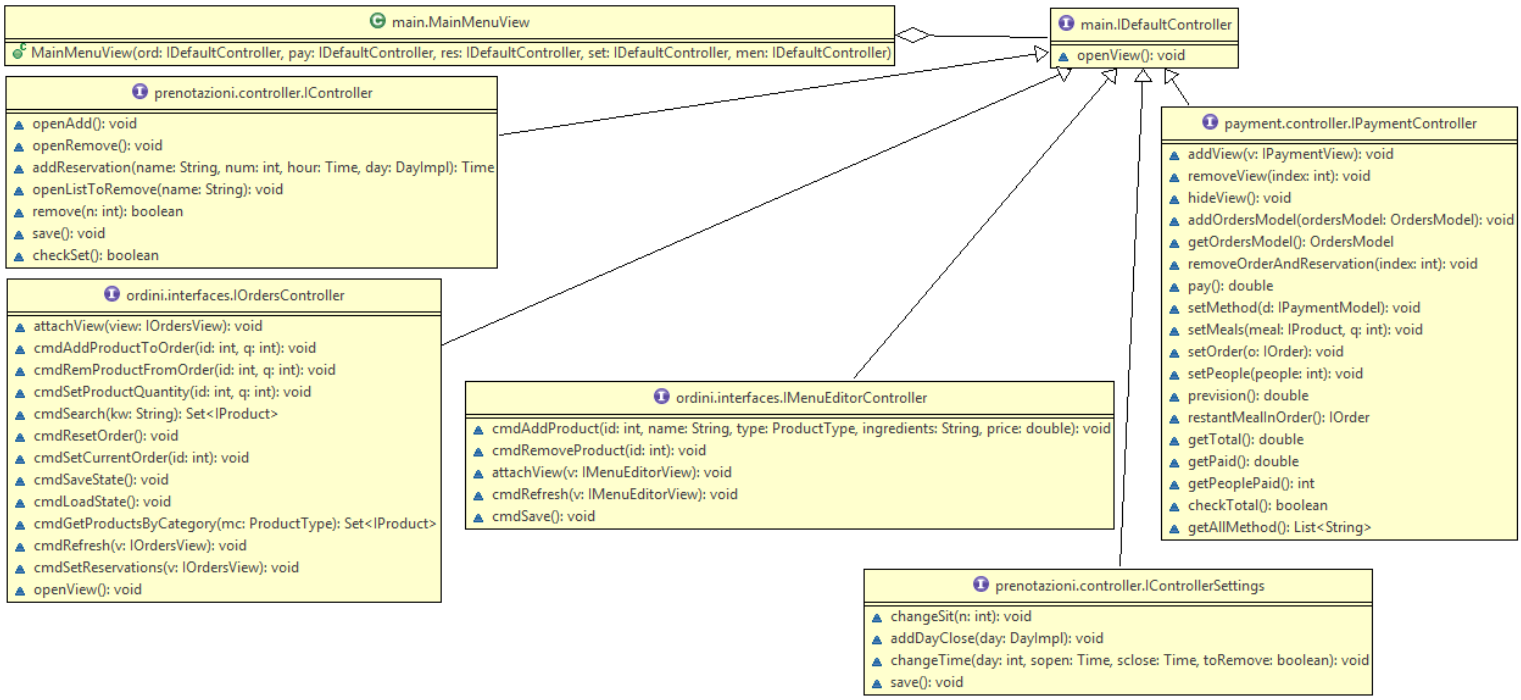
Un'altra problematica che non affronta il nostro elaborato è la gestione delle modifiche alle impostazioni e al menù durante l'attività. La modifica del menù mentre sono presenti degli ordini e la modifica dell'orario di apertura mentre sono presenti prenotazioni creano l'inconsistenza dei dati.

# Design

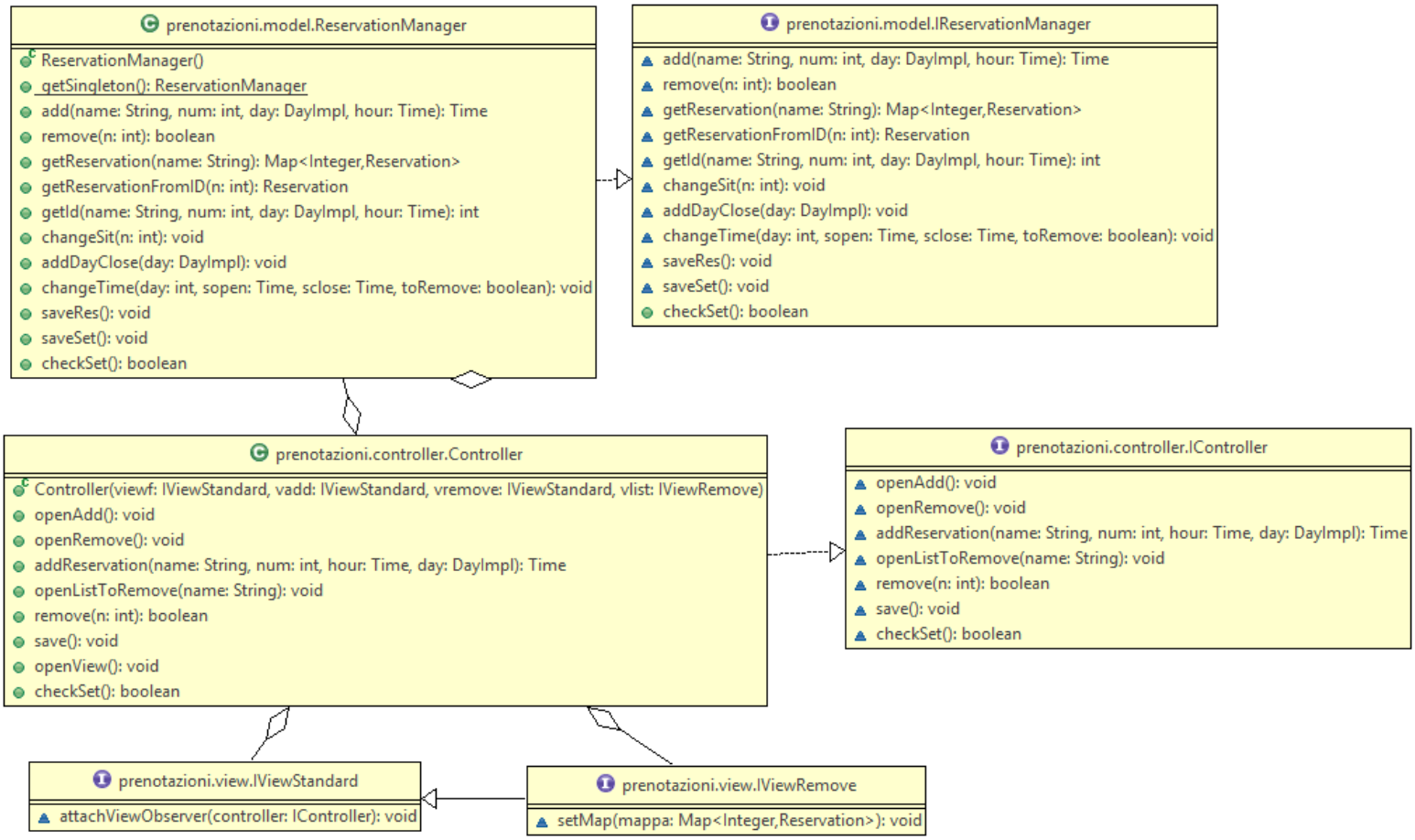
## 2.1 Architettura

Il programma è stato sviluppando seguendo il pattern MVC per rendere indipendente la parte grafica dalla parte che compone il nucleo dell'elaborazione.

Ognuna delle tre macro parti (prenotazioni, ordini, pagamenti) è sviluppata seguendo il pattern sopracitato. Vi è poi una parte che gestisce le interconnessioni tra di esse, come rappresentato dall'UML sottostante.



Per quanto riguarda le **prenotazioni** la parte del Model si occupa del calcolo della effettiva disponibilità del ristorante al momento della richiesta di una nuova prenotazione; e gestisce tutte le impostazioni relative al ristorante. Le Views sono diverse: una per aggiungere una nuova prenotazione, una per rimuovere e una per visualizzare le prenotazioni. Il Controller gestisce il collegamento tra le due parti. Come è illustrato dall'UML di seguito.

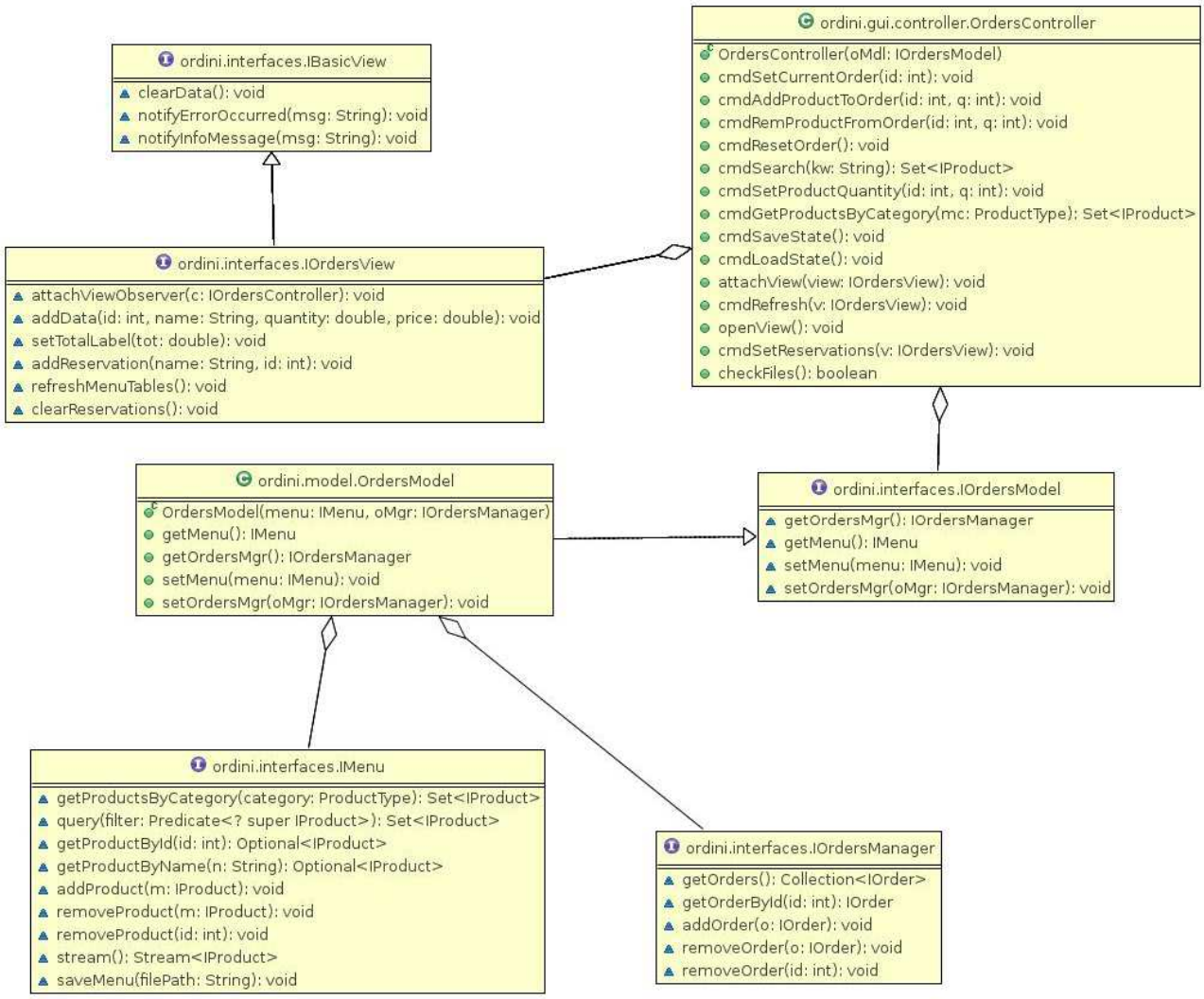


La parte della gestione delle **ordinazioni** comprende due parti:

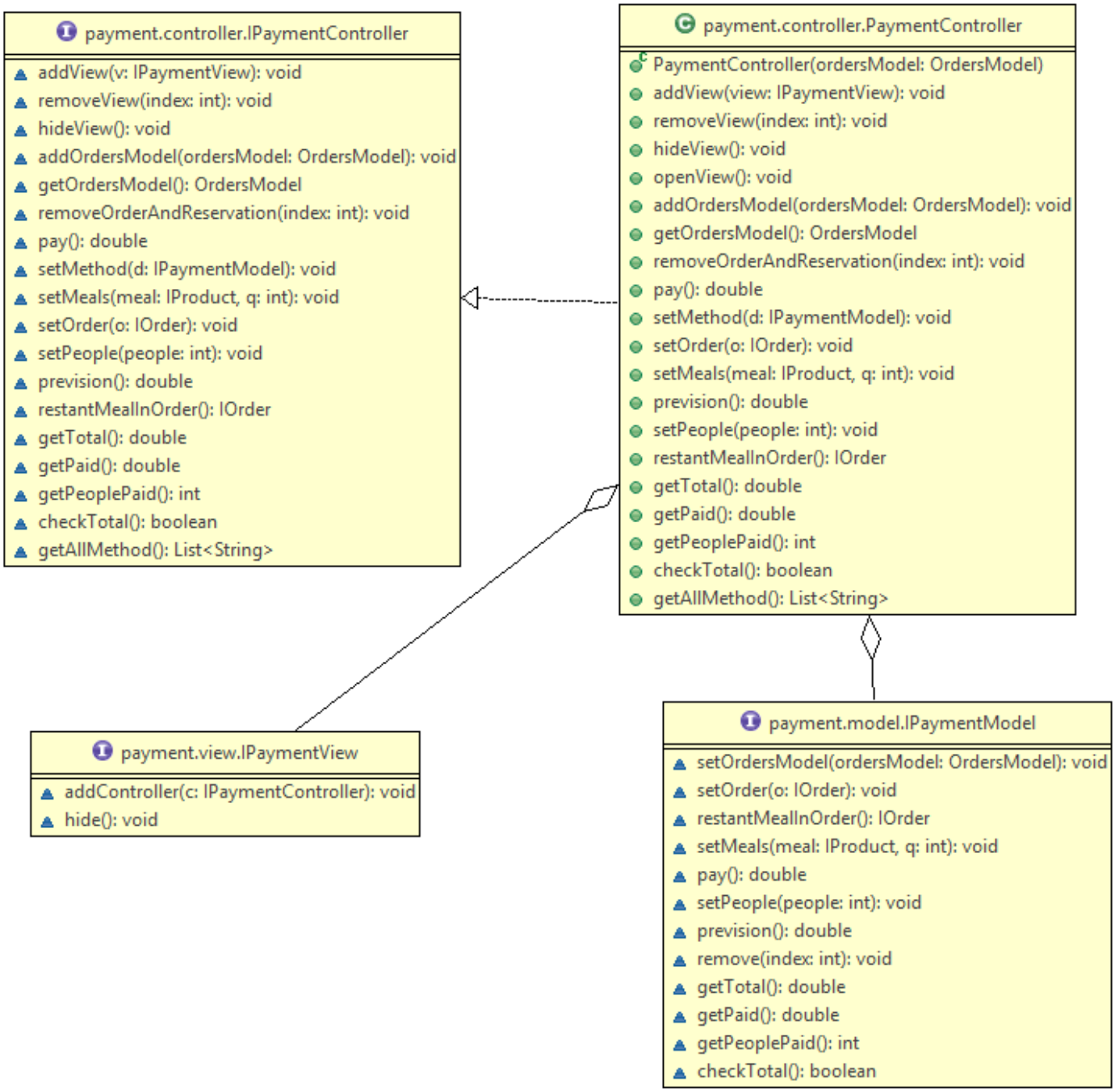
La prima parte si occupa della modellazione dei prodotti (piatti, bevande, ecc.) e la loro gestione attraverso un menu nel quale vengono catalogati in base a diverse categorie. Viene inoltre fornita all'utente un interfaccia grafica grazie alla quale è possibile modificare il menu aggiungendo e rimuovendo prodotti.

La seconda parte invece riguarda la modellazione degli ordini e la loro gestione, che avviene grazie a un 'gestore di ordinazioni' che ne regola l'aggiunta e la rimozione. L'interfaccia grafica fornita permette di:

- aggiungere o rimuovere un ordinazione dal sistema
- modificarne il contenuto
- ottenere il costo totale dei prodotti contenuti nell'ordine
- cercare un ordine esistente attraverso il suo numero identificativo



La View relativa alla parte di **pagamento** si occupa della selezione dell'ordine da evadere e della sua visualizzazione; e, infine, della selezione del metodo di pagamento. Il Model gestisce tutte le operazioni matematiche necessarie per l'evasione dell'ordine, partendo dalla selezione di quest'ultimo fino alla sua effettiva cancellazione, che avviene dopo il comando di pagamento effettuato. Il Controller, infine, si occupa della sincronizzazione tra le due parti.



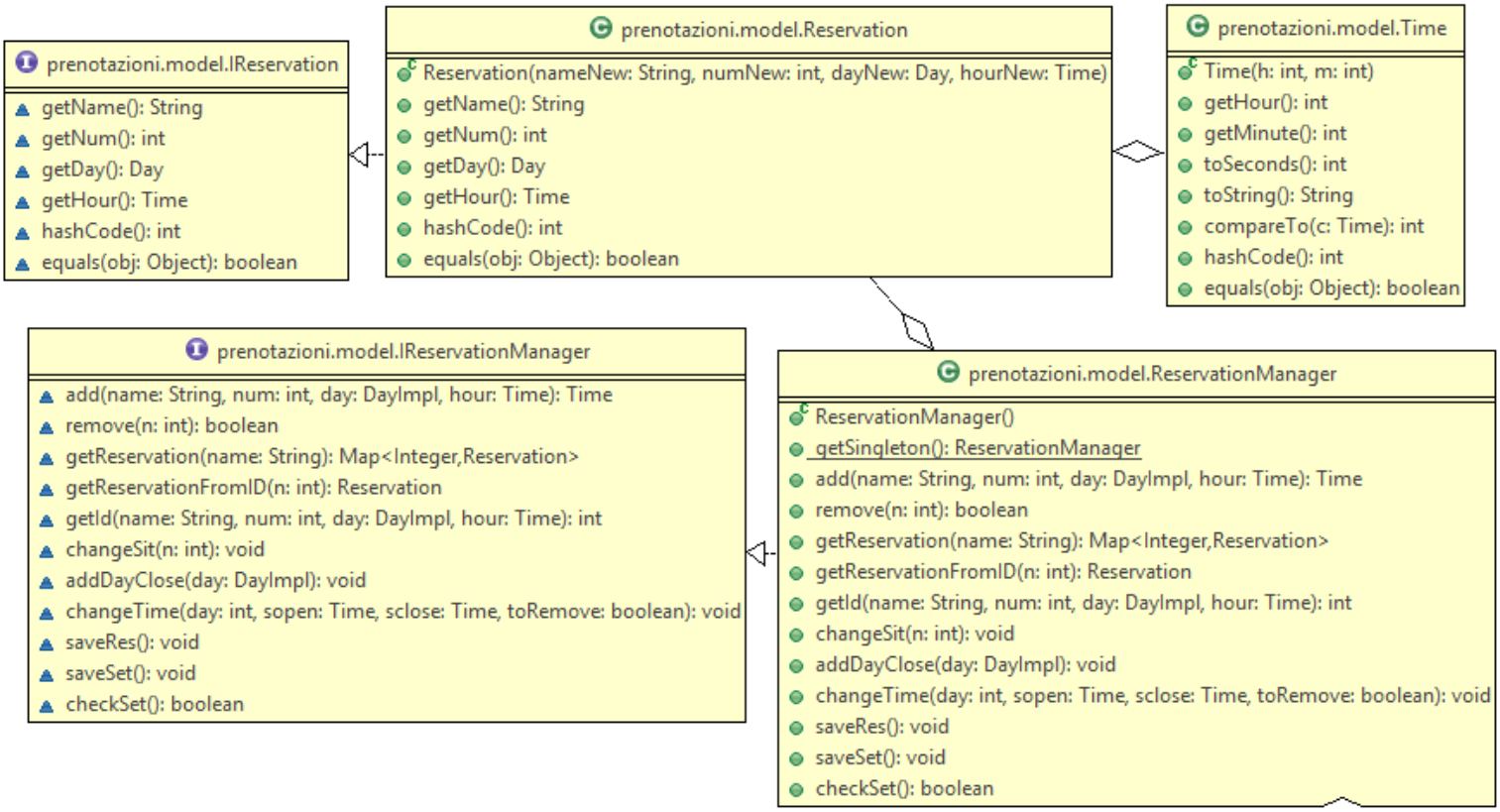
# 2.2 Design dettagliato

## Prenotazioni

Il model della parte di prenotazioni è composto dalla classe *ReservationManager* che implementa l'interfaccia *IRservationManager*.

Questa gestisce tutte le informazioni relative alle impostazioni del ristornate.

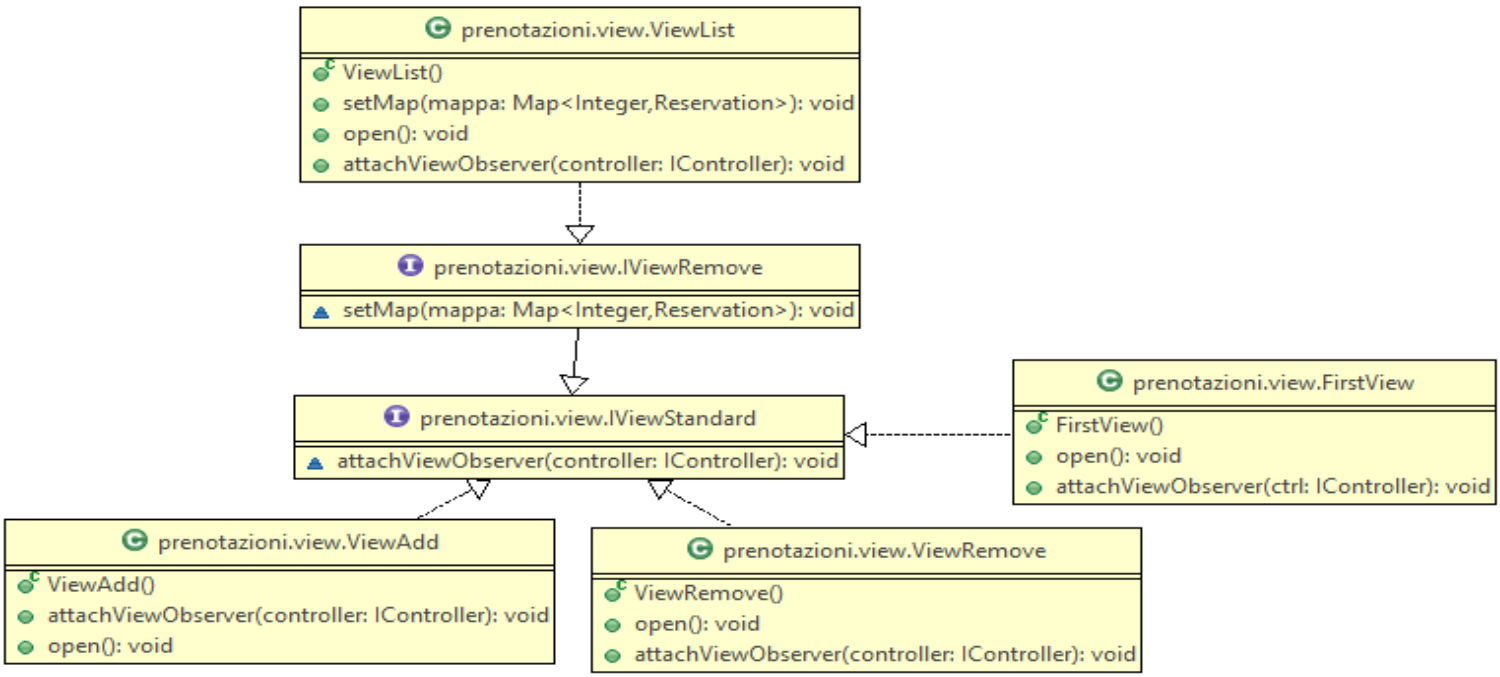
Ha quindi un campo che è una mappa di interi e *Reservation*, il primo è un ID univoco mentre il secondo è un oggetto della classe che implementa l'interfaccia *IReservationManager*. Quest'ultima rappresenta come è fatta una prenotazione, possiede quindi tutti i campi relativi alle informazioni di una prenotazione tra cui un oggetto della classe *Time*, classe creata per gestire un orario. L'ID univoco è però memorizzato solo nella classe *ReservationManager*. L'UML sottostante rappresenta questa composizione.



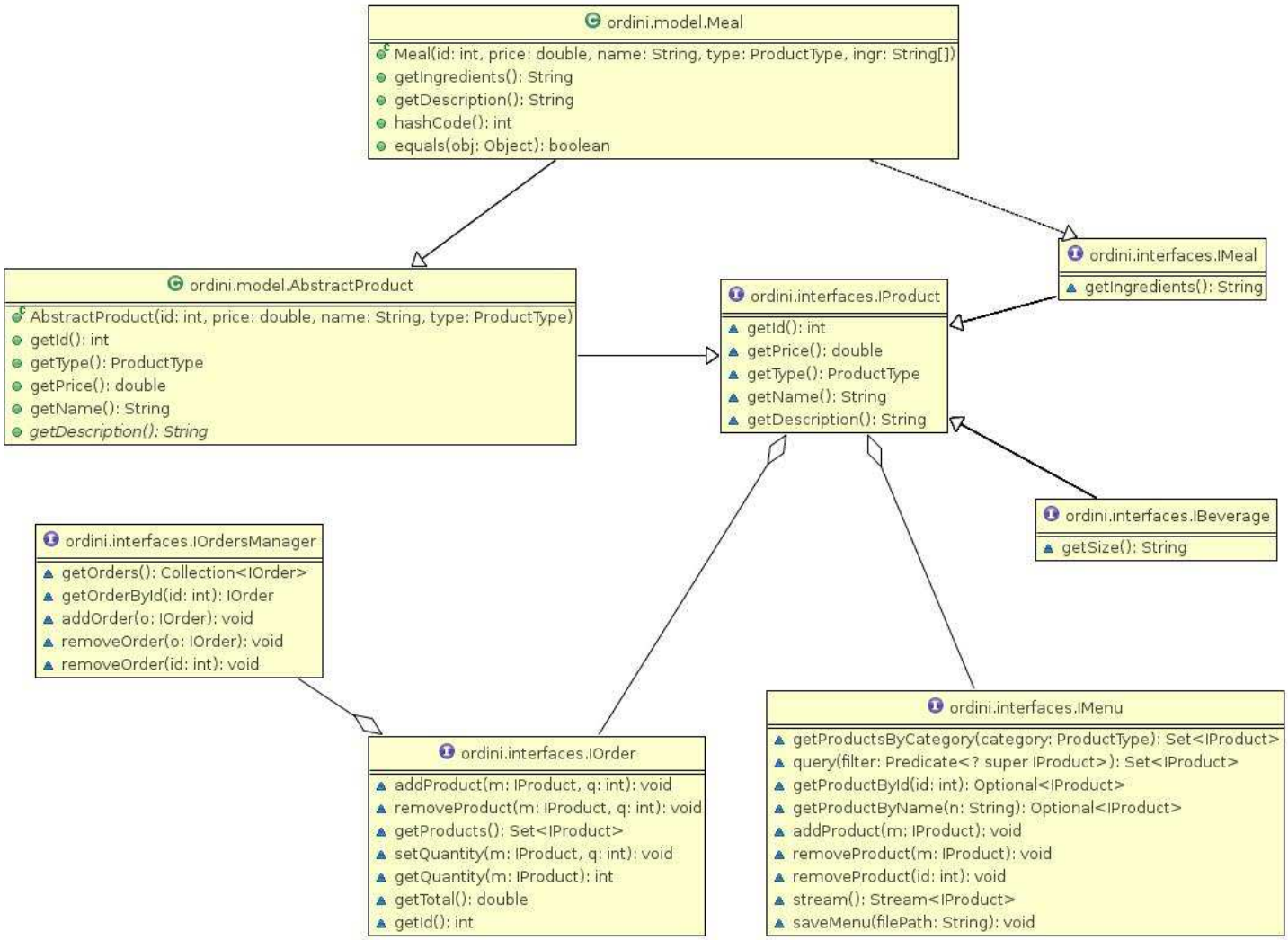
La classe *ReservationManager* ha quindi il metodo principale add utilizzato per aggiungere una prenotazione. Questo metodo richiama altri metodi privati che si occupano del calcolo dell'effettiva possibilità di aggiungere una prenotazione o meno. Possiede poi il metodo per rimuovere una prenotazione. Ha inoltre due metodi per ricercare le prenotazioni presenti; getReservation e getReservationFromID. Il primo prende come parametro una Stringa che è il nome a cui è salvata la prenotazione, siccome è prevedibile che questo non sia univoco viene restituita una mappa di prenotazioni. Il secondo invece prende come parametro tutti i dati di una prenotazione e restituisce il suo ID. Questo avviene ipotizzando che non esistono due prenotazioni con tutti i campi identici. La classe ha poi i metodi per gestire le impostazioni del ristorante.

La classe *ReservationManager* è implementato seguendo il pattern Singleton, poiché ho reputato che fosse necessario che all'interno del progetto vi fosse una sola istanza di questa classe.

Le Views per la gestione di tutte queste cose sono diverse. Tutte implementano l'interfaccia *IViewStandard*, tranne *ViewList* che implementa *IViewRemove*. Questa seconda interfaccia, comunque, estende la prima, poiché *ViewList* implementa tutti i metodi di *IViewStandard* però necessita anche di un metodo in cui settare la mappa di prenotazioni che si vuole stampare. L'UML sottostante rappresenta le Views.



## Ordini




*IProduct*: è l'interfaccia che modella un prodotto, definisce i metodi necesarri per esporne gli attributi che lo caratterizzano.

*AbstractProduct*: è una classe astratta che contiene un implementazione dei metodi comuni ad ogni prodotto definiti nell'interfaccia *IProduct*.

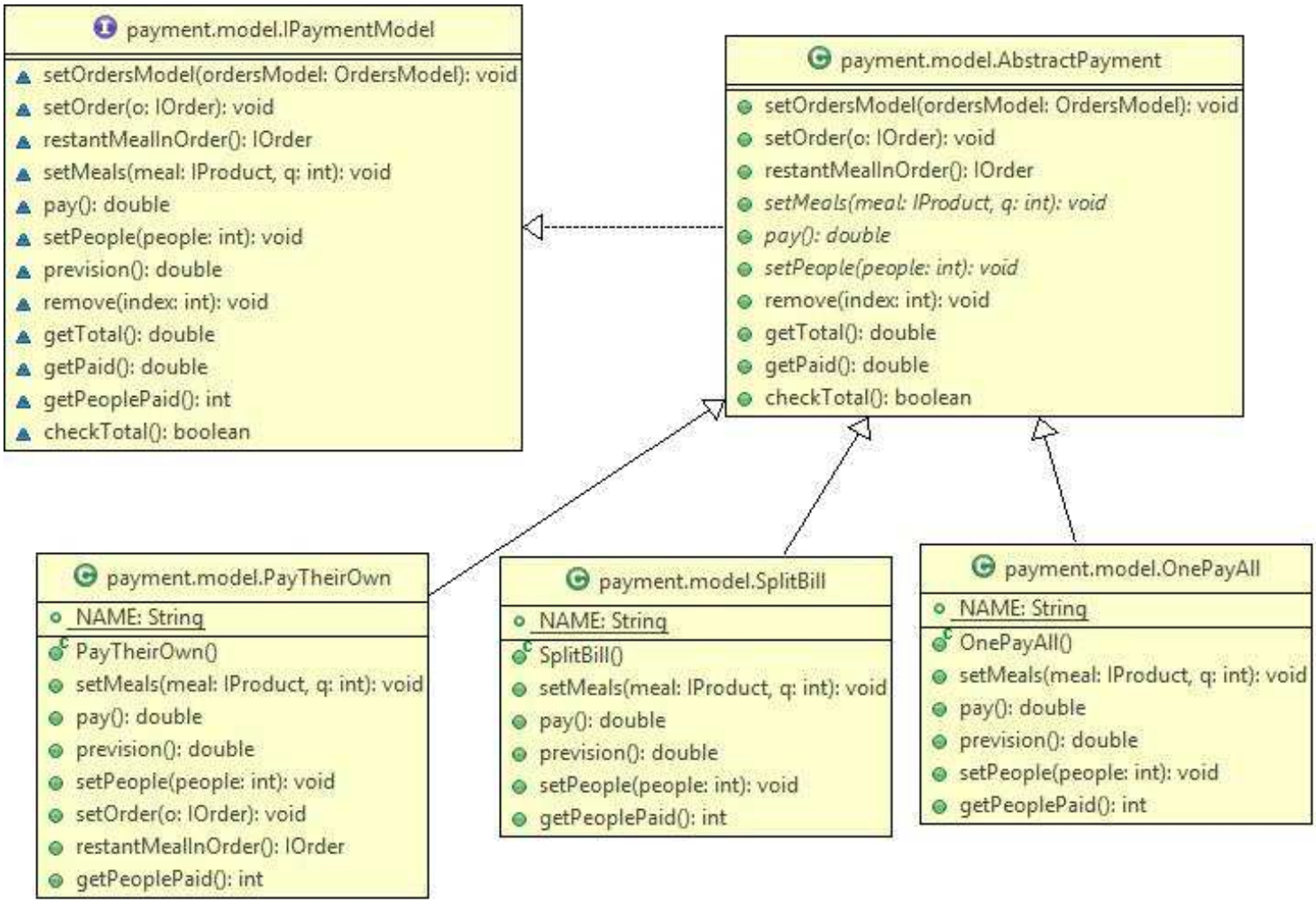
*IMeal* e *IProduct*: estendono l'interfaccia *IProduct* dalla quale ereditano i metodi generali per la gestione di ogni prodotto e aggiungono quelli necessari a descrivere il loro tipo più specifico, le classi *Meal* e *Product*, che rispettivamente implementano queste interfacce, estendono la classe *AbstractProduct* che fornisce già un implementazione dei metodi comuni ad ogni Product.

*Menu*/*IMenu*: modella il menu di un ristorante, fornendo una definizione dei metodi necessari alla gestione di un insieme di Prodotti per esempio aggiungere, rimuovere o effettuare ricerche sui prodotti contenuti nel menu.

*IOrder*: Modella l'ordinazione di un ristorante e definisce i metodi per modificarla e per ottenere informazioni sui prodotti in essa contenuti.

*IOrdersManager*: rappresenta il 'gestore degli ordini' e fornisce metodi per la gestione delle ordinazioni, che vengono univocamente identificate attraverso un id il quale coincide con quello della prenotazione relativa all'ordine stesso.

## Pagamento



La parte principale del pagamento riguarda la parte di calcolo.

Essendo i metodi di pagamento simili tra loro, ognuno di essi estende la classe astratta *AbstractPayment* che a sua volta implementa l'interfaccia *IPaymentModel* che contiene i principali metodi per lo svolgimento del pagamento cosi come *AbstractPayment* racchiude i principali elementi utili per effettuare i calcoli.

Ogni metodo di pagamento differisce dagli altri sostanzialmente per l'implementazione del metodo pay, in quanto ognuno di essi fornisce una diversa interpretazione di esso.

Nello specifico:

*OnePayAll* setta il campo relativo all'importo pagata uguale all'importo totale

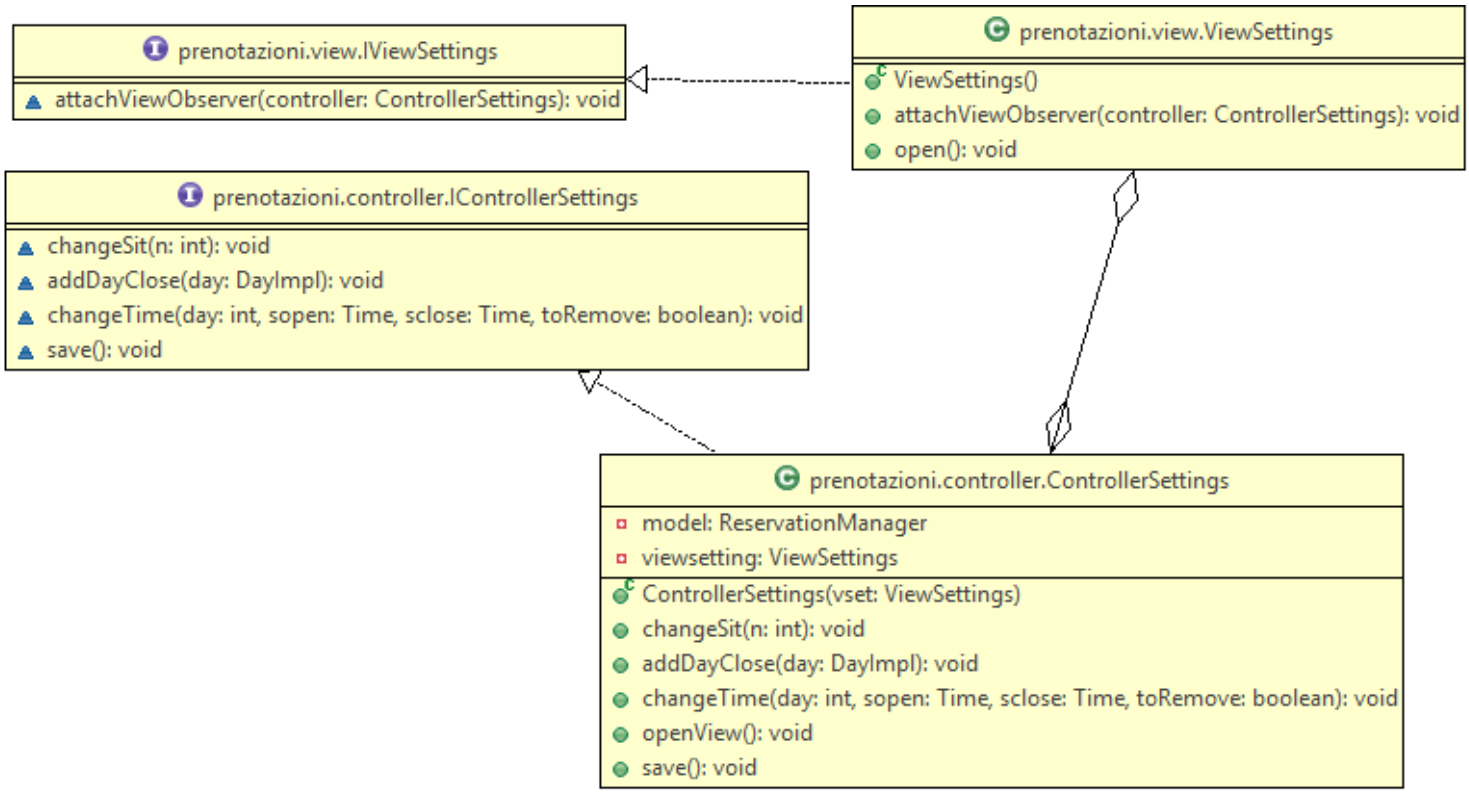
*SplitBill* incrementa il valore del campo paid di una somma data dalla divisione del totale per il numero di commensali arrotondato per difetto ai decimali tranne l'ultima sua chiamata prima del raggiungimento dell'uguaglianza tra totale e parziale, che risulta la differenza tra totale e pagato (rappresenta una quantità che non ha senso dividere tra i clienti ma che presa singolarmente ha valore, a compenso dell'arrotondamento).

*PayTheirOwn* somma al campo paid il valore della portata selezionato moltiplicato per la quantità selezionata.

I metodi estendono tutti la classe *AbstractPayment* perché presentano tratti comuni esistono comunque dei metodi specifici che vengono implementati da tutti ma chi non ne necessita lancia un'eccezione.

## Impostazioni generali

Per gestire le impostazioni vi è un apposita View chiamata *ViewSettings* che implementa *IViewSettings*, l'interfaccia non è la stessa delle altre View poiché il controller utilizzato è diverso e quindi il metodo attachViewObserver prende in ingresso un parametro diverso. L'UML sottostante rappresenta queste relazioni.




## Classi di utilità

Per quanto riguarda le classi di utilità utilizzate da tutti i membri del gruppo abbiamo:

Logger: questa classa implementa il pattern Singleton attraverso un enumeration con un unico elemento e fornisce la possibilità di tenere traccia di eventi avvenuti nell'applicazione tracciandoli su un file di log.

Config: è una classe di utility che permette di gestire un file di configurazione.

FileManager: fornisce funzionalità per la serializzazione di oggetti e il loro salvataggio e caricamento da file.

# Sviluppo

## 3.1 Testing automatizzato

Per verificare la correttezza del codice prodotto sono stati effettuati test automatizzati sulla parte sulla parte computazionale tramite JUnit, come illustrato a lezione. Nello specifico sono stati effettuati del test sui Models eseguendo i metodi e verificandone la correttezza del risultato step-by-step.

Relativamente alla parte grafica, sulla quale non abbiamo effettuato test automatizzati, abbiamo compensato effettuando test manuali sull'applicativo.

## 3.2 Divisione dei compiti e metodologia di lavoro

Il progetto è stato creato da un team di tre persone.

All'inizio del lavoro ci siamo ritrovati per decidere la suddivisione dei compiti. La suddivisione del programma è stata fatta come segue:

- Sara Sintoni: gestione delle prenotazioni e delle impostazioni generali del ristorante; con relativi controller, model e GUI.
- Matteo Venditto: gestione degli ordini e del menù; con relativi controller, model e GUI.
- Daniele Rosetti: gestione dei pagamenti; con relativi controller, model e GUI.
- Insieme: GUI iniziale e gestione dell'apertura delle varie parti di programma.

Dopo una prima parte di lavoro individuale, ci siamo resi conto che si erano create eccessive dipendenze tra le parti. Abbiamo cercato di eliminarle per quanto possibile, ma non siamo certi di aver raggiunto il risultato ottimale. Questa fase di correzione delle dipendenze è avvenuta tramite alcuni incontri dei membri del gruppo; lo sviluppo delle tre parti è però proseguito in modo individuale supportato dall'utilizzo di un repository Mercurial su Bitbucket.

Al termine della realizzazione delle tre parti ci siamo poi ritrovati per creare la parte iniziale comune. Questa parte è avvenuta senza eccessive complicazioni.

# 3.3 Note di sviluppo

Le classi del package *prenotazioni.giorni* sono state prese dalle soluzioni dell'esercizio 1 dell'appello 6 del 2013.

Le classi relative alla strategia utilizzata per il controllo dei dati inseriti nel form per l'aggiunta di un prodotto al menù sono state adattate dalla soluzione di un appello d'esame di OOP come indicato anche una documentazione delle suddette classi. Le classi sono: *IFormField, IFormStrategy, MenuEditorFormStrategy.*

Per effettuare l'arrotondamento dei valori nella classe *payment.model.SplitBill* è stato reperito un metodo su StackOverflow. Inoltre per aggiungere un immagine come background di un panel sono stati reperiti dalla medesima fonte. Questi metodi sono presenti nella classe *payment.view.SidePanel.*

# Commenti finali

## 4.1 Conclusione e lavori futuri

Terminato l'elaborato reputiamo che ci siano margini di miglioramento in esso.

Per quanto riguarda la parte delle prenotazioni si potrebbe suddividere la parte delle prenotazioni e quella delle impostazioni del ristorante, in modo da rendere il modello meno pesante. Gli algoritmi di calcolo per verificare l'effettiva possibilità potrebbero essere migliorati. Si potrebbero poi aggiungere controlli prima dell'eliminazione di una prenotazione controllando che non vi sia già un ordine associato.

Nella parte relativa agli ordini per migliorate la riusabilità del menù si vorrebbe aggiungere la possibilità di definire delle variazioni dei prodotti implementando questa funzione attraverso un pattern decorator.

Si vuole inoltre implementare la possibilità di modificare un prodotto già presente all'interno del menù.

Relativamente alla parte di pagamento si pensava di rendere il metodo di pagamento dividendo la spesa(Romana) per un numero diverso di commensali diverso dal totale riportato sull'ordine.

## 4.2 Difficoltà incontrate e commenti per i docenti

Da questa esperienza ne ricaviamo che il nostro approccio alla progettazione è stato, probabilmente, troppo superficiale. Da questo, deduciamo quindi, che in futuro sarebbe più opportuno dedicare più tempo alla progettazione prima di passare alla fase implementativa.

Abbiamo inoltre riscontrato alcune difficoltà nell'utilizzo del DVCS.

# Appendice A

# Guida utente

Insieme all’applicazione vengono forniti dei file per le impostazioni, che permettono di utilizzare subito il software. È previsto un menù e un orario di attività: tutti i giorni dalle 19 alle 24. Il progetto può essere avviato anche senza tali file, ma fino a quando le impostazioni e il menù non vengono settati, l’applicativo è inutilizzabile.

All’avvio dell’applicazione compare un’interfaccia grafica attraverso la quale è possibile scegliere tra le attività disponibili: prenotazioni, ordini, pagamento, impostazione, gestione menù.

È possibile scegliere tra inserire, rimuovere o visualizzare le prenotazioni tramite un interfaccia grafica con tre bottoni, l’aggiunta prevede l’inserimento di tutti i dati per poterla, dopodiché verrà visualizzato un messaggio con la risposta alla richiesta. Per rimuovere una prenotazione è sufficiente inserire il nome, poi verrà visualizzata una tabella con tutte le prenotazioni che corrispondo a quel nome. L’aggiunta di una prenotazione è possibile sempre, se però mancano delle impostazioni viene visualizzato un messaggio. È possibile comunque aggiungere, ma è molto probabile che l’aggiunta non vada a buon fine.

L’interfaccia grafica per la gestione degli ordini è composta da un'unica parte. Sono presenti due tabelle, una nella quale sono elencati i prodotti e l’altra che rappresenta l’ordine corrente, è possibile aggiungere piatti a quest’ultimo utilizzando la funzione di drag and drop e selezionandone la quantità che si desidera.

Per utilizzare la parte di pagamento è necessario selezionare innanzi tutto l’ordine da evadere, successivamente sarà possibile scegliere il metodo di pagamento desiderato che darà accesso a varie informazioni aggiuntive (es. le persone totali e quelle che hanno pagato nel pagamento alla Romana), se selezionato il metodo secondo il quale ognuno paga la propria consumazione è richiesto selezionare le portate dalla tabella. Una volta iniziato il pagamento di un conto non sarà possibile cambiare ne metodo ne ordine, al termine del pagamento l’ordine verrà evaso e sarà rimosso dal gestore degli ordini cosi come sarà rimossa la prenotazione.

Per quanto riguarda le impostazioni sia generali che del menù è possibile farle in ogni momento, è quindi necessario che l’utente prima di modificare il menù controlli che non ci siano ordini, e prima di modificare l’orario o il numero di posti controlli le prenotazioni già presenti.