



# BENNY'S PIZZA



Pisu Roberto, Giugliani Marco, Cristarelli Simone,  
Donati Giovanni, Nuvoli Andrea

21 Febbraio 2024

# Indice

1 Analisi	3
1.1 Requisiti . . . . .	3
1.2 Analisi e modello del dominio . . . . .	3
2 Design	5
2.1 Architettura . . . . .	5
2.2 Design dettagliato . . . . .	5
3 Sviluppo	13
3.1 Testing automatizzato . . . . .	13
3.2 Note di sviluppo . . . . .	13
4 Commenti finali	24
4.1 Autovalutazione e lavori futuri . . . . .	15
Guida utente	17

# Analisi

Il software ha come obiettivo quello di simulare la gestione di una pizzeria, sotto forma di un videogioco. Il gioco, che si ispira a “Good pizza, Great pizza!”, si basa sulla velocità dell’utente nel preparare correttamente le pizze richieste dai clienti.

## 1.1 Requisiti

### Requisiti funzionali

- L’utente che giocherà al videogioco interpreterà i ruoli di Benny: proprietario e pizzaiolo del suo locale “Benny’s Pizza”.
  - Il locale sarà diviso in due ambienti diversi:
    - o Sala: Benny interagirà con i clienti, i quali ordineranno una o più pizze (max 2) scegliendo tra quelle proposte nel menù del locale, attenderanno di ritirarle e pagheranno.
    - o Cucina: l’ambiente di lavoro di Benny. Il pizzaiolo ha il compito di eseguire le richieste dei clienti, preparando loro le pizze, partendo dalla stesura dell’impasto fino alla cottura di queste.
  - Il gioco si sviluppa su più giornate, in ognuna delle quali l’obiettivo di Benny sarà il raggiungimento di un determinato guadagno minimo entro la fine di ognuna.
  - Ogni giorno la difficoltà aumenta: aumenta il guadagno minimo da raggiungere. Benny dovrà quindi essere sempre più veloce a completare le pizze richieste.

### Requisiti non funzionali

- Benny dovrà essere il più efficiente possibile nell’utilizzo delle risorse: ingredienti. Dovrà anche essere il più preciso e pulito possibile per evitare di perdere tempo prezioso.

## 1.2 Analisi e modello del dominio

Il videogiocatore deve essere in grado di ricevere gli ordini da parte dei clienti preparando loro le pizze così come richieste.

I clienti sceglieranno una o più pizze da un menù rappresentato da un file. La richiesta avverrà tramite il nome della pizza e non tramite gli specifici ingredienti che la compongono. (Es: Margherita, Diavola, ecc...)

Ogni pizza deve essere univocamente riconoscibile dal suo nominativo e dovrà indicare un prezzo e un elenco di ingredienti.

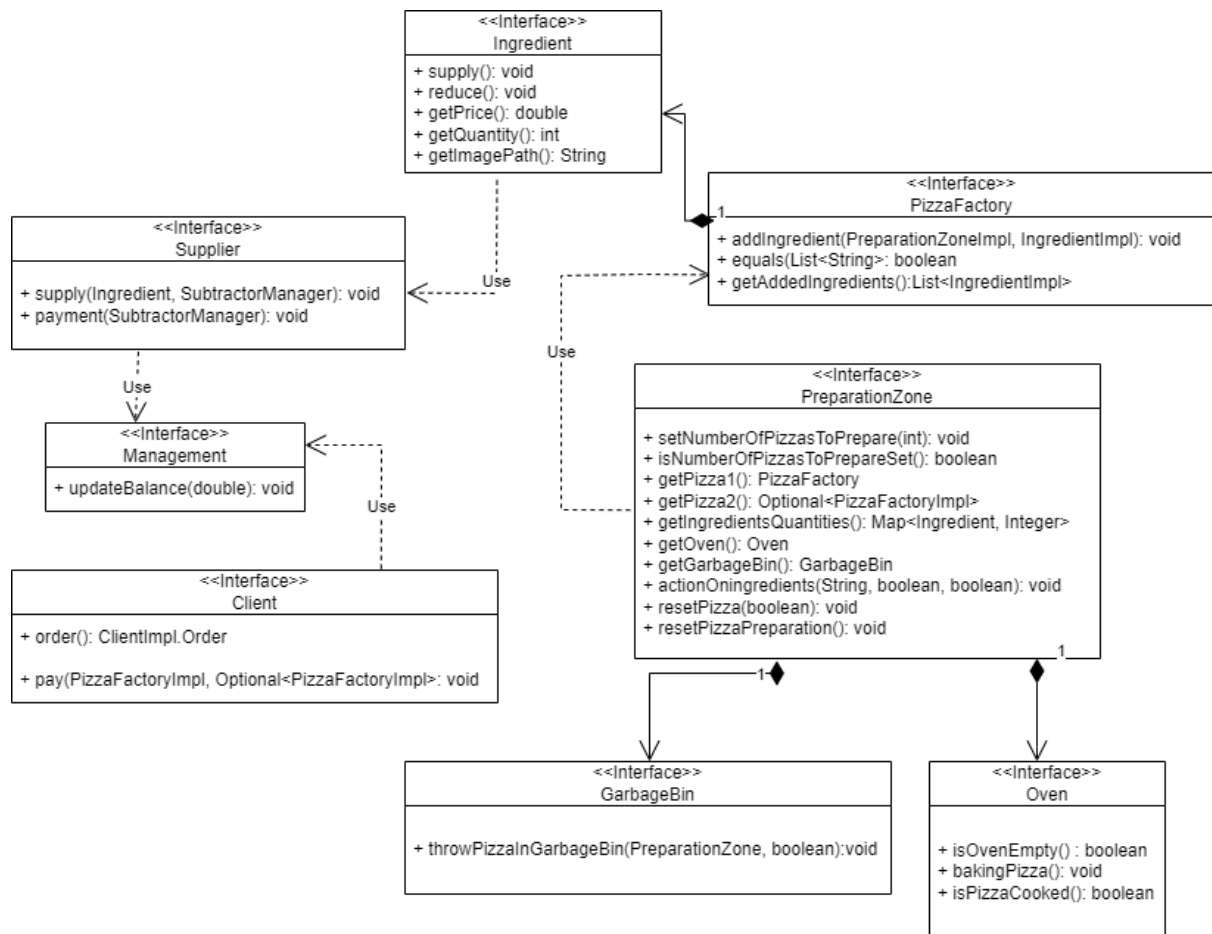
Il videogiocatore dovrà essere in grado di preparare le pizze aggiungendo i vari ingredienti partendo dalla base della pizza fino ai vari condimenti.

Si potrà anche buttare la pizza se non soddisfatti della preparazione (ingredienti sbagliati o omessi). Al termine della preparazione la pizza verrà messa in forno. Il gioco sarà in grado di gestire pagamenti in entrata derivanti dai clienti e in uscita per il rifornimento degli ingredienti.

Il pagamento in entrata dipende dal prezzo della pizza e da come viene realizzata, se fedele a come presentata nel menù il cliente paga il 100% del prezzo.

Nel caso in cui la preparazione della pizza risulti incompleta o sbagliata il cliente pagherà solamente per gli ingredienti aggiunti correttamente (i quali avranno un prezzo singolo); eventuali ingredienti in

più non verranno pagati dal cliente. A fine giornata, Benny controllerà il proprio guadagno per capire se la giornata può dirsi un successo oppure no.



# Design

## 2.1 Architettura

Per la realizzazione di “Benny’s Pizza” è stato adottato il pattern architetturale MVC (Model-View-Controller). In particolare, le interfacce del progetto sono divise nel seguente modo:

1. Il model si occupa di definire i vari elementi del gioco, in particolare pagamenti, clienti, tempo, posizionamento e rifornimento degli ingredienti, cottura ed eventuale cestinamento della pizza. Tutti questi elementi, come in una vera pizzeria, riescono a collaborare tra di loro condividendo componenti come, per esempio, l’ordinazione del cliente e la gestione dei soldi.
2. La view è un’interfaccia grafica reattiva realizzata con il framework di Java Swing e si compone di due schermate: una per la sala e una per la cucina.
3. Il controller mette in comunicazione model e view.

Utilizzando questo tipo di pattern architetturale permettiamo a view e model di rimanere separate e indipendenti, ma allo stesso tempo di comunicare tra di loro permettendo l’aggiornamento dell’interfaccia grafica nel momento in cui avviene una modifica ai campi del model.

## 2.2 Design dettagliato

### Andrea Nuvoli

*Implementazione del menù della pizzeria.*

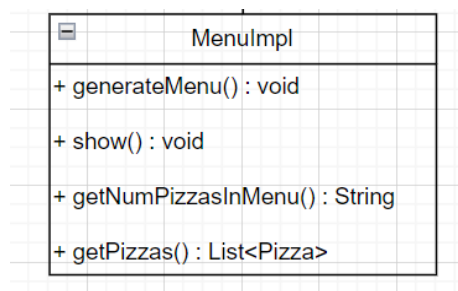
**Problema:** Implementare il menù della pizzeria estraendolo da un file.

**Soluzione:** La classe ‘Menu’ viene intesa come classe helper, la quale è utile esclusivamente al cliente, che, arrivato in pizzeria, procederà ad effettuare l’ordine scegliendo tra le pizze messe a disposizione.

Realizzazione di un file JSON: il file è stato utilizzato per contenere tutte le pizze del menu sotto forma di lista di oggetti della classe Pizza, aggiunta come classe innestata statica all’interno proprio della classe ‘Menu’, che si occupa dell’implementazione del suddetto.

Utilizzo della libreria Jackson per il parsing del file: attraverso di un oggetto ObjectMapper è stato possibile leggere il file specificando il percorso assoluto (stando attenti a mantenere la proprietà di portabilità del codice) deserializzando il contenuto del file JSON in una lista di oggetti Pizza.

Per quanto riguarda la parte grafica, il menu è visibile all’interno di una OptionDialog a seguito di un click sull’apposito bottone.

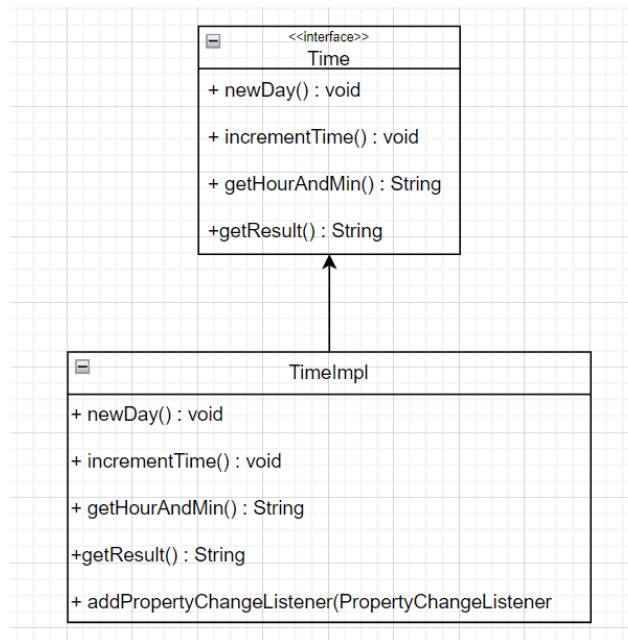


*Gestione del tempo.*

**Problema:** Realizzare un orologio virtuale che simula lo scorrere della giornata lavorativa.

Mantenere il numero di giorni che si possono dire essere superati con successo.

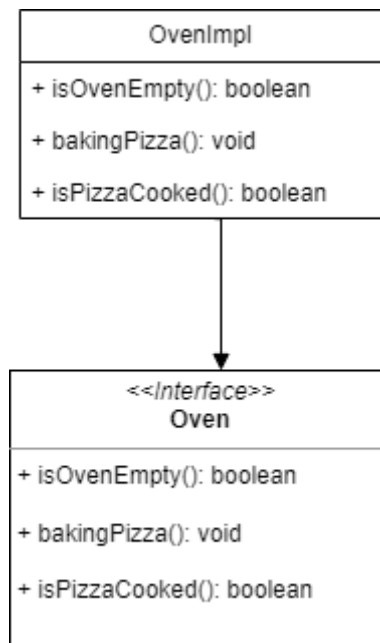
**Soluzione:** La simulazione del tempo avviene attraverso l'utilizzo di un oggetto della classe Timer. Infatti, sfruttando il metodo schedule() si è programmato l'aggiornamento dei campi della classe. In particolare, ogni 5 secondi reali nel gioco passeranno 15 minuti. Al termine della giornata avviene un controllo sul guadagno effettuato e se risulta sufficiente la giornata può dirsi superata



*Parte della grafica della sala della pizzeria in collaborazione con Pisu Roberto.*

## Donati Giovanni

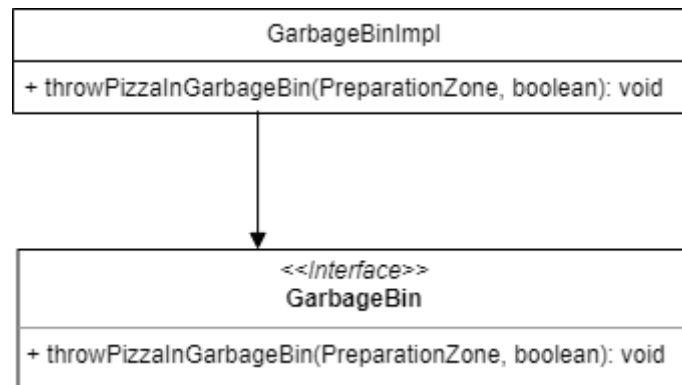
*Implementazione del forno della pizzeria.*



**Problema:** Implementare una simulazione di un forno dove il passare del tempo è indice della cottura della pizza.

**Soluzione:** La simulazione del tempo che passa avviene attraverso l'uso di un oggetto della classe Timer e tramite il metodo `scheduleAtFixedRate()`. Al termine di un tot di secondi (5 nel gioco, che simulano i minuti che passa una pizza in forno) viene cambiato lo stato della pizza che passa da false (cruda) a True (cotta). Con il metodo appena citato, nella parte di grafica si verificavano diversi problemi, si è optato quindi, al fronte di diversi tentativi, di far passare il tempo tramite una `Thread.sleep()`. Come argomento viene passato il tempo, in millisecondi (1500 nel gioco) che, nonostante blocchi tutto per quel lasso, è ragionevolmente breve da non far percepire che tutto sia fermo. Al termine della `Thread.sleep()` le pizze sono cotte ed è possibile quindi consegnarle al cliente.

*Cestino*

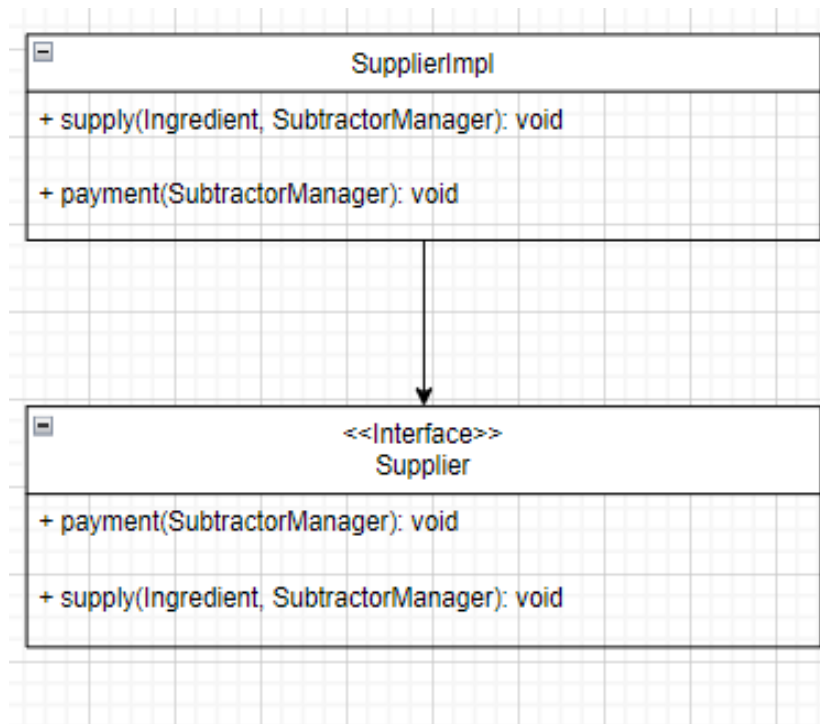


**Problema:** Buttare la pizza se condita male, o per qualsiasi altro motivo.

**Soluzione:** Il cestino chiama praticamente il metodo `resetPizza()` della classe `PreparationZone`. Il metodo chiamato cancella la pizza.

**Cristarelli Simone**

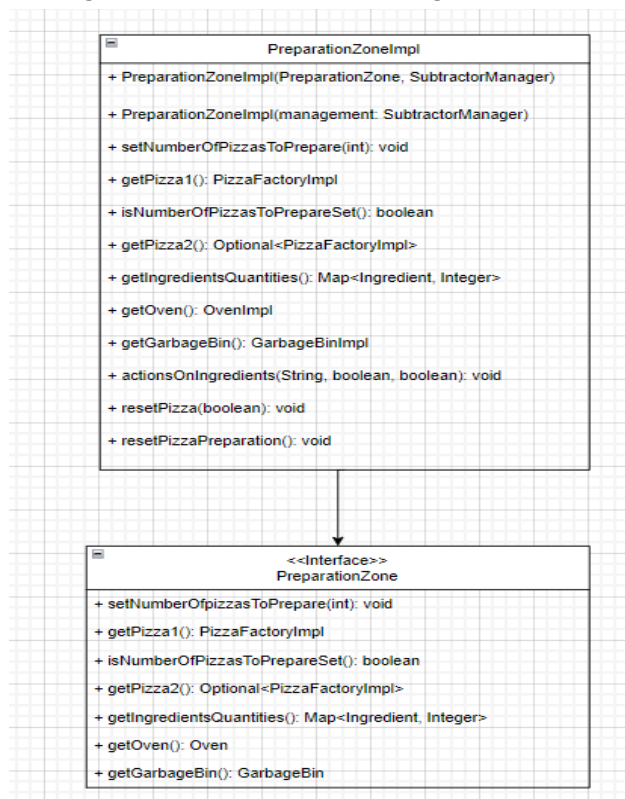
*Implementazione dei rifornimenti.*



**Problema:** Implementare i rifornimenti degli ingredienti. Una volta avvenuto il rifornimento bisogna togliere dei soldi dal bilancio.

**Soluzione:** Utilizzo di un apposito SubtractorManager per far effettuare il pagamento all'utente e del metodo supply() presente in ogni ingrediente per rifornirli.

*Implementazione della pizza e gestione del bancone con Giugliani Marco.*



**Problema:** Implementare un bancone (`PreparationZone`) per gestire i vari elementi della cucina che poi si andranno a unire, nel controller, con gli elementi del salone.

**Soluzione:** La gestione del bancone è avvenuta grazie alle varie classi che rappresentano ciò che effettivamente si riesce a trovare nella zona della cucina sia sfruttando metodi interni a quelle classi sia implementando nuovi metodi che aiutano nella gestione generale, come ad esempio `actionsOnIngredients()` che gestisce l'aggiunta e il rifornimento degli ingredienti. Sono stati anche gestiti gli ingredienti in una mappa insieme alle proprie quantità così da facilitare eventuali controlli su queste ultime anche nel controller.

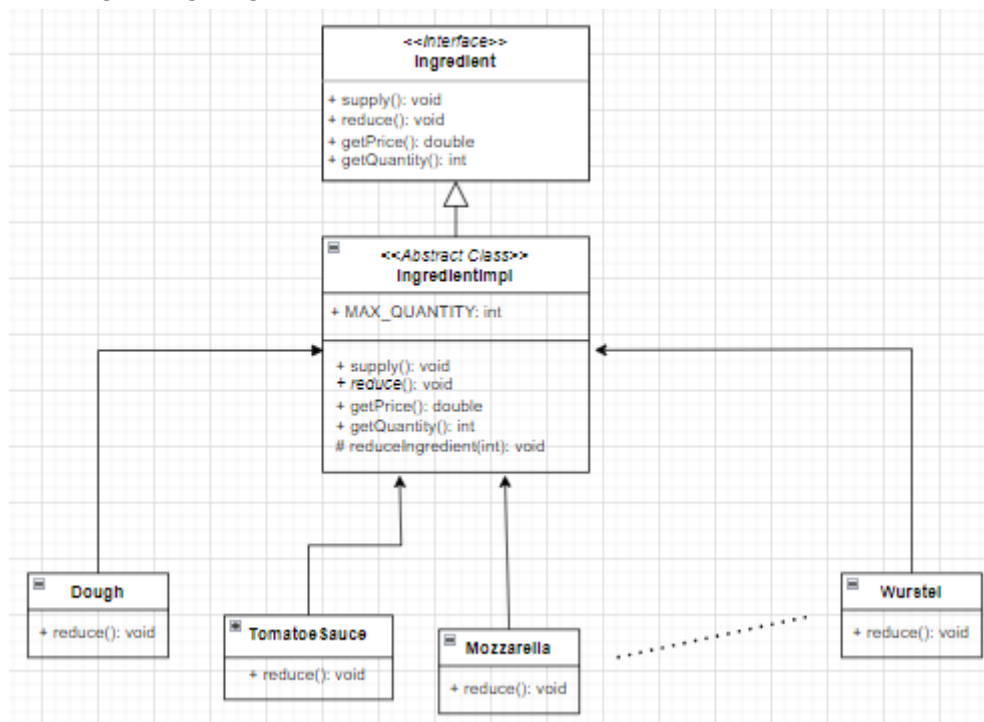
*Parte della gestione del controller.*

*Parte della gestione della grafica della zona cucina.*



## Giugliani Marco

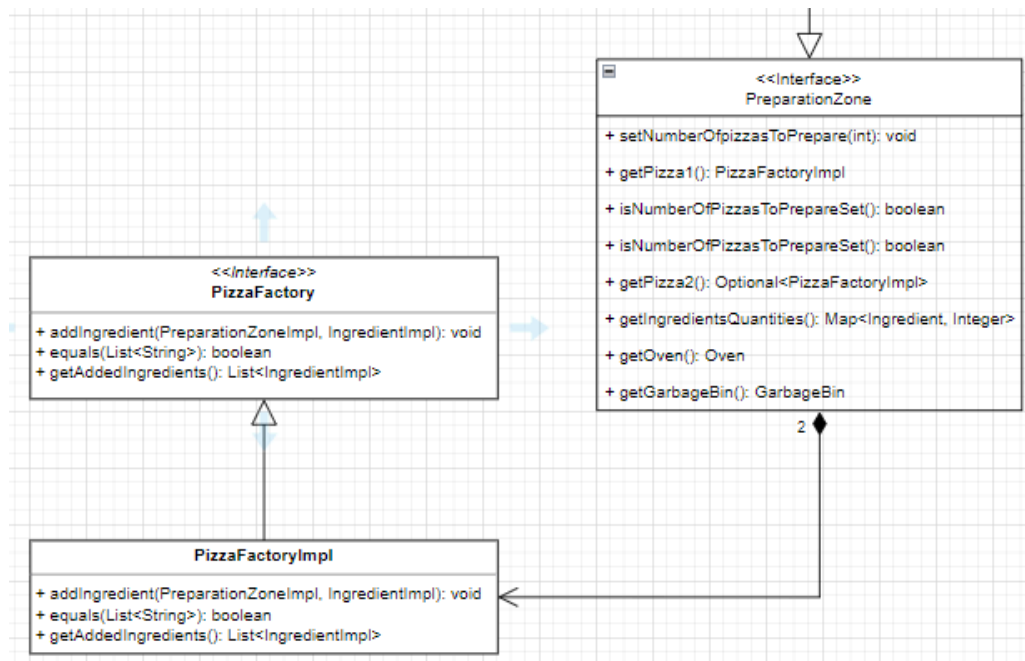
*Gestione della logica degli ingredienti*



**Problema:** Implementare la logica degli ingredienti, organizzandoli in classi evitando il più possibile ripetizioni di codice uguale o molto simile.

**Soluzione:** Implementata una sorta di “Template Method”, in cui le classi dei vari ingredienti a disposizione (ognuna contenente dati diversi, riguardanti, ad esempio, la quantità da scalare ogni qual volta questi vengono messi sulla pizza, o il singolo prezzo), ereditano da una classe astratta 'IngredientImpl', contenente tutti i metodi comuni a qualunque ingrediente (quali per esempio la riduzione o il rifornimento). L'unico metodo la cui implementazione è diversa per ciascuno di essi, è proprio quello della 'reduce()', il quale dipende, come detto in precedenza, dalla quantità da scalare quando viene farcita la pizza in fase di preparazione. La diversa quantità da scalare viene gestita dal metodo protected 'ReduceIngredient(int)'.

*Preparazione della pizza (in collaborazione con Cristarelli Simone)*



**Problema:** Implementare logicamente e graficamente la preparazione di una pizza in cucina, cercando di piazzare gli ingredienti richiesti dal cliente di turno.

**Soluzione:** Implementata l'interfaccia (con conseguente classe che la implementa) PizzaFactory, la quale consente, di volta in volta, di selezionare un ingrediente e di metterlo sulla pizza (e di conseguenza ridurre la sua quantità), facendo inoltre vari controlli fondamentali, come per esempio che prima di tutti venga messo l'impasto.

Contiene anche, e soprattutto, il metodo 'equals()', che consente, nella fase di pagamento da parte del cliente, di determinare se la pizza preparata risponde alle esigenze di quest'ultimo, situazione fondamentale per la quantità di denaro che poi entrerà nella cassa della pizzeria.

Questo metodo, per determinare se le pizze sono uguali, fa un semplice confronto fra gli ingredienti richiesti e quelli effettivamente usati. Questo è stato possibile tramite l'override, nella classe astratta 'IngredientImpl', del metodo 'equals()' della classe 'Object'.

Gli oggetti PizzaFactory, 2 in questa versione del gioco, sono contenuti in un oggetto PreparationZone (interfaccia descritta da Cristarelli Simone).

*Sviluppo parziale della grafica della cucina (in collaborazione con Cristarelli Simone e Donati Giovanni)*

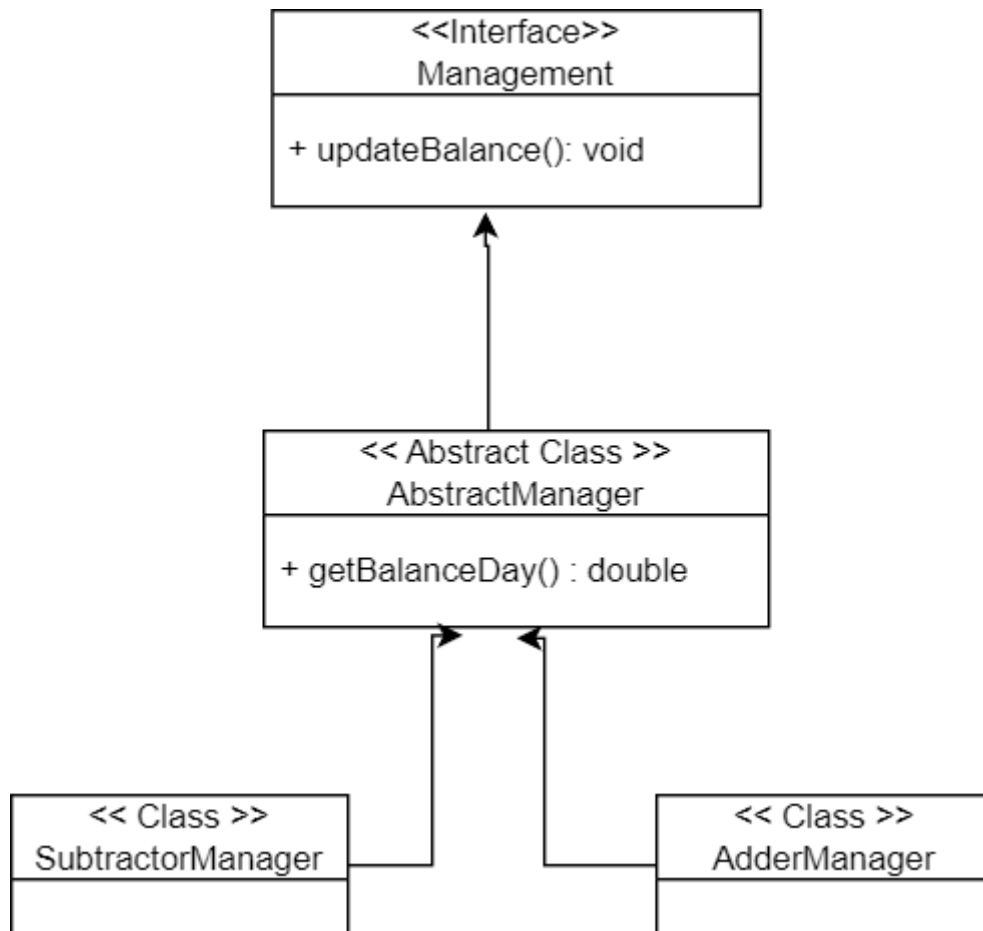
## Pisu Roberto

### Gestione della logica dei pagamenti

**Problema:** Implementare la logica dei pagamenti, in particolare suddividere il gestore delle entrate da quello delle uscite, per evitare che la stessa classe mi gestisca entrambe.

**Soluzione:** Tramite una struttura gerarchica ho creato una classe astratta che contenesse il campo del saldo da incrementare/diminuire e 2 classi figlie che implementano quest'ultima ognuna in modo differente (ognuna effettua i dovuti controlli prima di effettuare l'operazione di aggiornamento del saldo), una che incrementa solamente il saldo e l'altra che lo diminuisce.

In questo modo, laddove è necessario aggiungere soldi al saldo, si verrà in contatto solamente con la classe AdderManager, mentre laddove è necessario sottrarre soldi al saldo, si verrà in contatto solamente con SubtractorManager.



#### *Gestione del cliente*

**Problema:** Implementare la logica del cliente che arriva, effettua l'ordine, aspetta che la pizza sia pronta e dopo di che paga il prezzo della pizza pieno se la pizza fornitagli contiene tutti gli ingredienti della pizza richiesta, altrimenti paga solamente per gli ingredienti che sono stati messi correttamente.

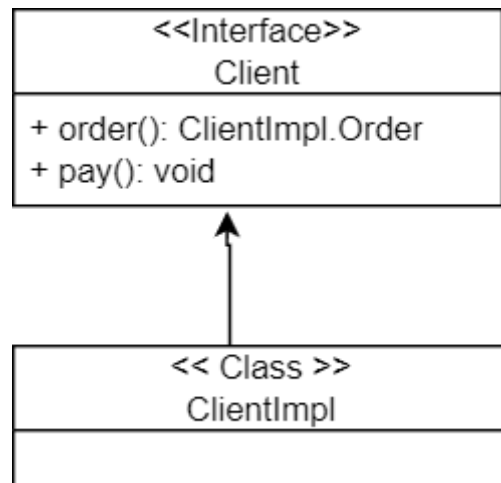
Inoltre va simulato la fila di clienti che arrivano uno dopo l'altro e che effettuano ordinazioni diverse.

**Soluzione:** Il cliente, oltre che all'interfaccia che lo rappresenta e alla sua implementazione, è implementato anche attraverso una classe innestata all'interno della classe ClientImpl per gestire l'ordine, (quest'ultima si chiama infatti Ordine).

Per quanto riguarda la simulazione della fila di clienti, viene implementata tramite un thread cliente all'interno del controller, che, una volta effettuato l'ordine va in waiting e aspetta che il pizzaiolo (colui che gioca) lo sveglia.

Una volta sveglia, il cliente effettua il pagamento in base alla pizza che gli viene fornita dal pizzaiolo.

Per mostrare l'ordine della pizza nell'interfaccia grafica è stato necessario l'utilizzo di un altro thread che si sincronizza (sempre tramite un gioco di wait e signal) con il thread cliente per mostrare alla gui l'ordine solamente dopo che il thread cliente ha effettivamente effettuato l'ordine.



*Parte della grafica della sala della pizzeria in collaborazione con Nuvoli Andrea.*

# Sviluppo

## 3.1 Testing automatizzato

Per il testing automatizzato abbiamo utilizzato JUnit nella versione 5.10.1

Ogni interfaccia implementata possiede il suo corrispettivo test, tranne nelle classi che utilizzano elaborazioni randomiche al loro interno, come ad esempio la classe ClientImpl la quale contiene i metodi pay e order entrambi con un fattore randomico e quindi di difficile testing.

Stessa cosa per il Supplier perché utilizza metodi di altre classi e che quindi sono testati direttamente nei loro test e per il TimeImpl, il quale, lavorando con un thread è stato testato mediante un main temporaneo che ne verifica il corretto funzionamento.

## 3.2 Note di sviluppo

### Roberto Pisu:

Utilizzo di Optional. Un esempio è il seguente:

Permalink :

<https://github.com/Imcloudss/OOP23-bennys-pizza/blob/452308a2f76ef6324e890f7f69db5af5407dc856/src/main/java/it/unibo/model/impl/ClientImpl.java#L30>

Utilizzo di lambda expression:

<https://github.com/Imcloudss/OOP23-bennys-pizza/blob/452308a2f76ef6324e890f7f69db5af5407dc856/src/main/java/it/unibo/view/UpdateThread.java#L39>

Utilizzo e sincronizzazione tra thread

Permalink dei 2 thread sincronizzati tra loro:

<https://github.com/Imcloudss/OOP23-bennys-pizza/blob/452308a2f76ef6324e890f7f69db5af5407dc856/src/main/java/it/unibo/controller/impl/ClientThread.java#L42-L56>

<https://github.com/Imcloudss/OOP23-bennys-pizza/blob/452308a2f76ef6324e890f7f69db5af5407dc856/src/main/java/it/unibo/view/GUIHallImpl.java#L393-L406>

### Simone Cristarelli:

Utilizzo di Optional. Un esempio è il seguente:

Permalink :

<https://github.com/Imcloudss/OOP23-bennys-pizza/blob/0bfd55b523f49dea915175468e4484c5abf31223/src/main/java/it/unibo/model/impl/PreparationZoneImpl.java#L118>

Utilizzo di Stream e di lambda expressions. Un esempio è il seguente:

Permalink :

<https://github.com/Imcloudss/OOP23-bennys-pizza/blob/0bfd55b523f49dea915175468e4484c5abf31223/src/main/java/it/unibo/model/impl/PreparationZoneImpl.java#L153-L155>

### **Andrea Nuvoli:**

Utilizzo della libreria esterna Jackson per estrarre dati da un file JSON, segue un esempio.

Permalink:

<https://github.com/Imcloudss/OOP23-bennys-pizza/blob/452308a2f76ef6324e890f7f69db5af5407dc856/src/main/java/it/unibo/model/impl/MenuImpl.java#L33>

Utilizzo dei thread per eseguire un metodo ogni 5 secondi evitando il busy waiting , segue un esempio.

Permalink:

<https://github.com/Imcloudss/OOP23-bennys-pizza/blob/452308a2f76ef6324e890f7f69db5af5407dc856/src/main/java/it/unibo/model/impl/TimeImpl.java#L61-L70>

### **Marco Giugliani:**

Utilizzo degli Stream, segue un esempio:

<https://github.com/Imcloudss/OOP23-bennys-pizza/blob/452308a2f76ef6324e890f7f69db5af5407dc856/src/main/java/it/unibo/model/impl/PizzaFactoryImpl.java#L60-L62>

Utilizzo di oggetti di tipo Optional, segue un esempio:

<https://github.com/Imcloudss/OOP23-bennys-pizza/blob/452308a2f76ef6324e890f7f69db5af5407dc856/src/main/java/it/unibo/model/impl/PreparationZoneImpl.java#L184>

Utilizzo della Reflection:

<https://github.com/Imcloudss/OOP23-bennys-pizza/blob/452308a2f76ef6324e890f7f69db5af5407dc856/src/main/java/it/unibo/model/impl/PreparationZoneImpl.java#L70-L73>

Utilizzo di lambda expression, qui un esempio:

<https://github.com/Imcloudss/OOP23-bennys-pizza/blob/452308a2f76ef6324e890f7f69db5af5407dc856/src/main/java/it/unibo/model/impl/PizzaFactoryImpl.java#L56>

### **Giovanni Donati:**

utilizzo di un Thread:

<https://github.com/Imcloudss/OOP23-bennys-pizza/blob/452308a2f76ef6324e890f7f69db5af5407dc856/src/main/java/it/unibo/model/impl/OvenImpl.java#L62-L65>

# Commenti finali

## 4.1 Autovalutazione e lavori futuri

### **Pisu Roberto:**

In mia opinione il gruppo in linea di massima ha lavorato bene, in armonia, aiutandoci l'un l'altro laddove ne avevamo bisogno, personalmente credo di essermi impegnato al massimo dando sempre il 100%.

Ho avuto un po' di difficoltà nel lavorare con i thread, nonostante le nozioni fornitemi anche dal corso di sistemi operativi, ma alla fine ne sono uscito credo vincitore.

Per essere la prima volta a lavorare ad un progetto di sviluppo informatico a gruppi mi sono trovato bene nel collaborare e condividere il progetto mediante git.

Sicuramente potevamo gestire meglio dall'inizio i tempi e gli impegni di ciascun componente del gruppo per poter lavorare meglio, ma in fin dei conti mi ritengo abbastanza soddisfatto.

### **Cristarelli Simone:**

Personalmente credo che come gruppo abbiamo lavorato bene, e anche come singolo credo di aver contribuito abbastanza per cercare di fare un buon progetto. A livello personale credo che all'interno del gruppo siamo quasi tutti sullo stesso livello anche se magari qualcuno spicca un pochino di più e per questo non mi ritengo il migliore ma neanche il peggiore del gruppo, sono abbastanza convinto che le mie conoscenze siano ad un livello buono e quindi credo di aver comunque dato un'ottima mano. Ciò che credo sia il mio punto di forza maggiore è quello di trovare spesso un modo di risolvere i problemi in maniera semplice anche se a volte potrei avere delle lacune su determinati argomenti, mentre sicuramente faccio più fatica nella parte di controllo dei thread, proprio durante questo progetto mi sono accorto di avere delle difficoltà in questo. Credo comunque che questo progetto sia servito a farci migliorare su molti punti di vista, sia a livello di conoscenze sia a livello personale perché ci ha dato la possibilità di solidificare ciò che abbiamo appreso a lezione e ci ha dato la possibilità di capire come si lavora in gruppo. Se dovessi portare avanti il progetto cercherei di migliorare di sicuro alcuni problemi di gestione dei thread che ha il progetto e cercherei anche di migliorare la grafica, poi si potrebbe ampliare il progetto con una gestione più ampia del gioco magari gestendo imprevisti o magari con più pizze.

### **Nuvoli Andrea:**

Implementare un videogioco all'interno un gruppo di cinque persone è stata un'esperienza entusiasmante e formativa. Mi ha permesso di apprezzare la complessità e la creatività coinvolte nello sviluppo di un prodotto interattivo. Durante il processo, abbiamo affrontato alti e bassi nel lavoro di gruppo: momenti di collaborazione e ispirazione alternati a sfide e conflitti nel coordinamento delle attività. È stato di cruciale importanza imparare a comunicare efficacemente e a organizzare le nostre risorse per massimizzare l'efficienza.

Questo progetto per il corso di OOP ci ha fornito una preziosa esperienza pratica nell'applicare le nostre conoscenze teoriche, preparandoci per le sfide future nel mondo professionale dell'informatica.

**Giugliani Marco:**

Per quanto riguarda il mio lavoro, posso ritenermi abbastanza soddisfatto, sia per come ho gestito le dosi di lavoro, sia per il codice prodotto, anche se, come in tutte le cose, ci sono stati alcuni alti e bassi, soprattutto a livello di organizzazione.

Però, lo sviluppo di questo progetto ci sarà molto d'aiuto per il futuro, avendo vissuto sulla pelle per la prima volta l'esperienza di programmare e coordinare un progetto di gruppo e di sicuro impareremo a gestire meglio il tutto, imparando dagli errori commessi durante questo periodo di tempo. Tornando a me, credo che la parte in cui ho trovato più difficoltà sia stata la programmazione della GUI, dove sicuramente avrò tempo per migliorare, mentre credo che nella parte di model abbia dato il meglio a mia disposizione in questo momento.

Per quanto riguarda una possibile futura continuazione del progetto, le idee che avrei in mente sono sicuramente di aumentare il numero di pizze che possono essere richieste da un cliente, ma anche un miglioramento significativo della grafica.

**Donati Giovanni:**

Credo che il gruppo abbia lavorato bene, ad ogni difficoltà c'è sempre stato qualcuno disposto a dare una mano e a chiarire eventuali problemi. Per quanto riguarda il mio lavoro mi ritengo soddisfatto il giusto anche se sono ben consapevole che potevo fare molto di più. Non ho trovato qualcosa di più facile o più difficile in quanto mi sono reso conto che tra tutti, per via di problemi di terze parti, ero il meno preparato sulla programmazione ad oggetti. Mi è stata quindi molto di aiuto la grande mano datami dai miei colleghi, sempre disponibili. Credo che questo progetto sia servito per consolidare i rapporti tra i componenti del gruppo, ma anche a fare fronte a diversi problemi organizzativi riscontrati. È stato utile per imparare anche ad ascoltare i propri collaboratori. Sicuramente, in un futuro, un eventuale lavoro di gruppo sarà più facile da affrontare avendo vissuto questa esperienza. Sono fiducioso che in futuro, acquisendo sempre più conoscenza, questo progetto, possa avere una primavera, possa essere migliorato e reso grande.

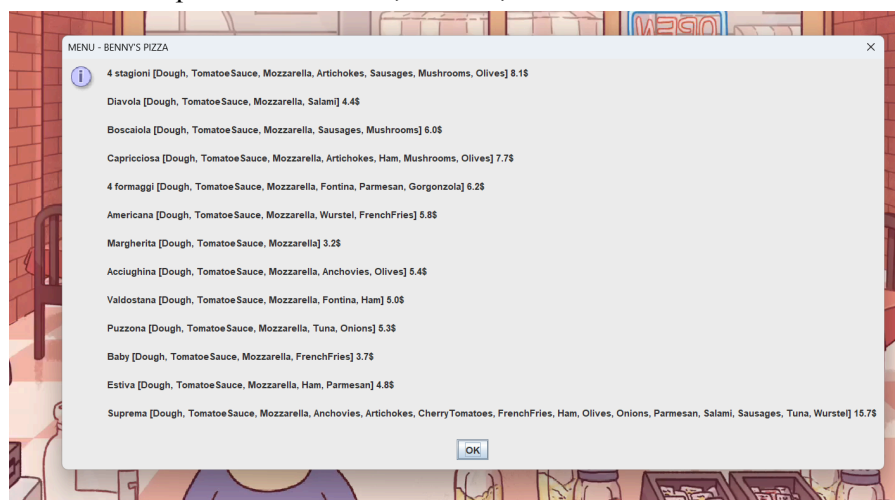


# Guida utente

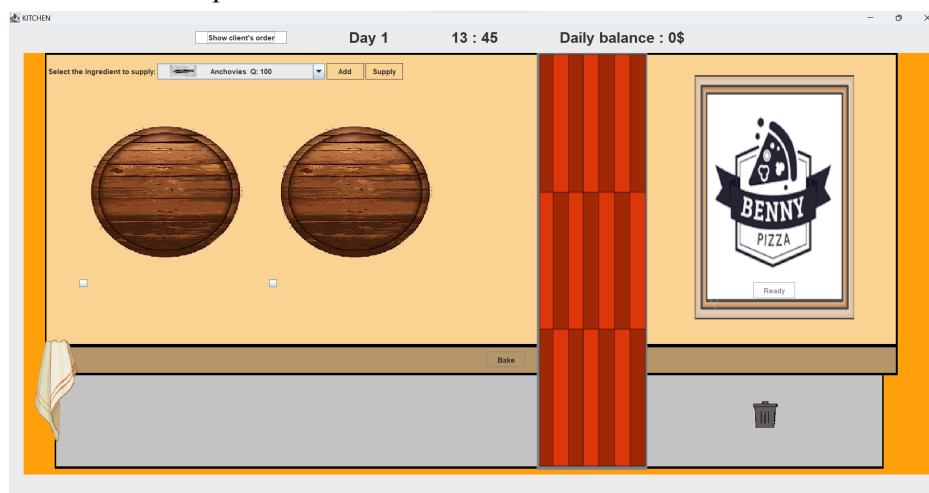
Aperta l'applicazione l'utente si impersonifica nel pizzaiolo, si trova quindi davanti un cliente, il quale ordinerà una o due pizze. Gli ordini verranno visualizzati a video da un popup.



In basso a destra si trova il pulsante menù che, cliccato, mostra tutto il menù




Chiudendo il popup dell'ordine, tramite la X in alto a destra del popup, si viene mandati alla schermata del bancone della pizza.



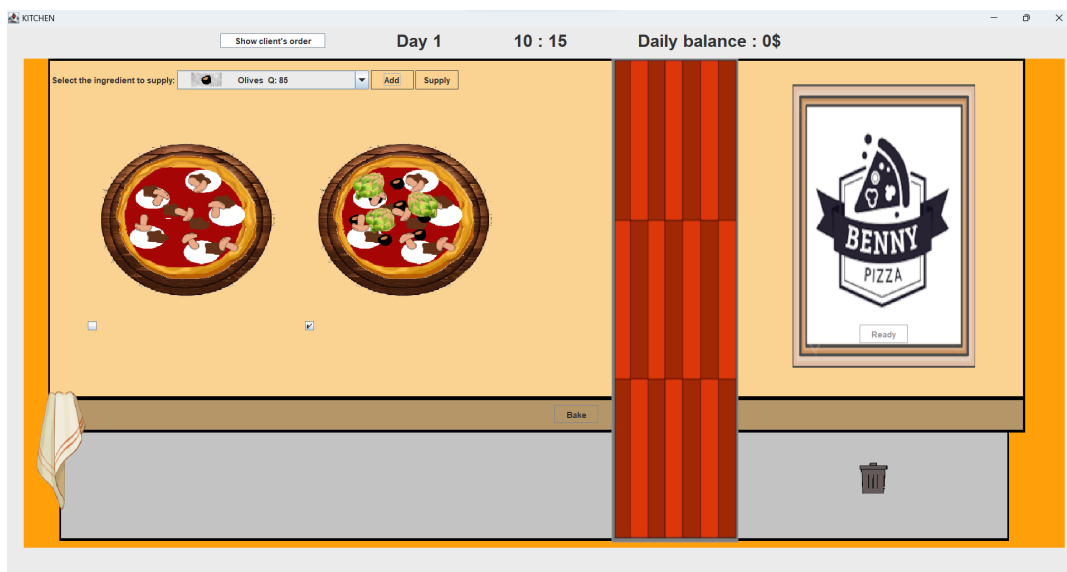
Nella finestra del bancone è possibile visualizzare l'ordine del cliente cliccando il bottone "show client order".

Per preparare la pizza, è bene prima spuntare il checkbox sotto il tagliere su dove si vuole mettere la pizza, quindi si seleziona un ingrediente dal listbox e premere "add" per aggiungere l'ingrediente.



Se si sbaglia il condimento o si vuole semplicemente buttare la pizza, tenendo selezionato il checkbox della pizza e cliccando sul bottone "cestino" in basso a destra  si butta la pizza.

Una volta completate le pizze, cliccando sul pulsante "bake" le pizze vengono messe in forno una di seguito all'altra.



Quando le pizze saranno cotte, il bottone "ready" verrà abilitato e, una volta cliccato, riporterà alla schermata iniziale, ricevendo quindi una nuova ordinazione.

Alla fine della giornata, se il guadagno è stato maggiore di un certo ammontare, l'utente ha vinto!