Progetto Metodologie di Programmazione

Studenti che hanno realizzato il progetto:

Daniel Zanchi email: daniel.zanchi@stud.unifi.it;
Daniele Landi email: daniele.landi1@stud.unifi.it;
Federico Guerri email: federico.guerri@stud.unifi.it.

Gli studenti devono sostenere anche la prova scritta.

Descrizione del progetto

Abbiamo scelto di sviluppare la traccia del self-bar, sostituendolo però con una pizzeria. L'applicazione è in grado di proporre al cliente tre diversi casi base,(per ogni tipologia di piatto: pizze, focacce, bevande) per poi eventualmente arricchirli con degli ingredienti a scelta. Il conto può essere saldato tramite tre diversi metodi di pagamento: contanti, bancomat oppure mobile. L'applicazione è stata pensata per essere utilizzata in un tablet.

Scelte di design principali

Durante la realizzazione del progetto abbiamo cercato di specializzare il più possibile le varie classi, facendo in modo che ognuna potesse avere un numero minimo di funzionalità. Questa scelta è dovuta al fatto di voler rendere il codice più facilmente modificabile. Abbiamo, sempre per questo motivo, creato numerose interfacce, che sono state implementate dalle diverse classi che compongono il progetto.

- Abbiamo deciso di utilizzare il design pattern Decorator, al fine di risolvere il problema principale, ossia poter agire su un oggetto in modo da aggiungere informazioni di vario tipo. Infatti tramite il pattern menzionato, è possibile arricchire alcuni casi base ("BasicWhitePizza", caso base della pizza) con delle informazioni contenute in altre classi, definite come toppings ("Anchovy",arricchimento per tutte le classi che estendono "Pizza").

Nei seguenti esempi verrà considerato soltanto il decorator utilizzato per gli oggetti di tipo pizza, dato che il funzionamento risulta essere il medesimo per beverage e focaccia.
Abbiamo creato una classe astratta "Pizza" che viene estesa dalla classe, anch'essa astratta, "PizzaDecorator" e dai tre casi base "BasicBrownPizza", "BasicDoubleWhitePizza" e "BasicWhitePizza". Quindi abbiamo fatto uso della classe "PizzaDecorator", estendendola in tutte le classi di "tipo" topping, in modo da poter implementare i metodi astratti "getInfo" e "getPrice". Questi ultimi restituiscono, rispettivamente, la descrizione del caso base attualmente considerato, arricchita con l'informazione del topping scelto ed il suo prezzo.

- Abbiamo deciso di utilizzare il design pattern Strategy, per poter distinguere fra loro le varie modalità di pagamento: "Cash"; "Bancomat"; "Mobile Payment".
Vengono utilizzate tre strategie, appunto, ognuna per un diverso metodo di pagamento. A seconda della strategia scelta, vengono aggiunti ad un pannello specifico, diversi componenti. Ad esempio,nel pagamento con contanti si visualizza un tastierino numerico, anziché due menù a tendina per scegliere l'operatore che effettua il pagamento da smartphone ("Mobile Payment").
Per attuare il pattern si utilizza l'interfaccia "PayMethodInterface", che specifica il metodo "payMethod", sviluppato in modi differenti all'interno delle varie classi che la implementano. La strategia scelta viene impostata tramite la classe "PayChooser", di cui viene istanziato un oggetto al momento del pagamento. In seguito verrà quindi richiamato il metodo "payMethod" della strategia impostata nella classe "PayChooser", in modo da creare i componenti necessari per poter eseguire la tipologia di pagamento scelta.

Esecuzione tipo del programma

Il programma viene eseguito dalla classe “Main”, che crea un nuovo oggetto della classe Pizzeriadiddieffe. In quest'ultima vengono creati: il frame che ospiterà tutti i pannelli dell'applicazione(metodo "initialize"); il pannello che ospiterà la prima schermata, "chooseNumberOfTables". Tale pannello viene quindi impostato come principale all'interno della finestra "frame"; le quattro etichette e i quattro pulsanti, utili per impostare il numero di tavoli interni ed esterni. Per fare questo vengono usati i metodi "createFormattedButton" e "createFormattedLabel", che utilizzano l'oggetto "myComponentCreator" per creare ed impostare le proprietà desiderate del componente in questione; infine vengono creati i tre pannelli, che rispettivamente, si riferiscono: alla scelta dell'azione da compiere "orderPanel"(l'utente può scegliere se ordinare dei cibi, modificare l'ordine o pagare il conto); ai due pannelli che ospitano i tavoli, interni ed esterni. Il primo è un'istanza della classe "JPanelWithBackgroundOrder", in grado di gestire l'ordine considerato in un dato momento. Gli ultimi due invece sono oggetti della classe "JpanelWithBackgroundTables", utilizzata per rappresentare i vari tavoli nei luoghi "interno" ed "esterno"; quindi vengono impostati i pannelli ai quali i tasti "back", delle rispettive schermate, si riferiscono. A tale scopo viene richiamato il metodo(ereditato) "setVisiblePanel" della classe "JPanelWithBackgroundTabels", in grado di impostare un determinato pannello,come punto di "ritorno" del tasto "Back" presente nel pannello considerato.

Una volta impostato il numero di tavoli, è sufficiente premere il tasto "Done", per accedere al pannello di scelta, che visualizza due pulsanti. Il primo di essi permette di accedere al pannello che mostra i tavoli interni, il secondo quelli esterni.
Cliccando quindi, per esempio, sul pulsante "inside", verrà mostrato una schermata contenente il numero di tavoli scelto in precedenza. Ogni tavolo è istanza della classe "Table", ed è caratterizzato da un "id" univoco. Quest'ultimo viene utilizzato per associare ad ogni tavolo un diverso ordine.
Cliccando quindi su un "tavolo", si accede alla prossima schermata.

Il pannello attuale, che riporta in alto il titolo "Order Manager", è in realtà, un oggetto di tipo "JpanelWithBackgroundOrder"(creato precedentemente, nella classe pizzeriadiddieffe, con il nome di "orderPanel").
Al suo interno sono presenti quattro pulsanti, ognuno dei quali permette una diversa azione. Inoltre ognuno di essi è istanza della classe "JbuttonTextImage", utilizzata per creare dei pulsanti con un'immagine di sfondo e del testo al centro.
Ogni ordine è istanza della classe "Order" e viene creato, oppure restituito dall'oggetto "myOrderManager" (istanza dell'omonima classe). Tale oggetto si occupa infatti, tramite l'"id" del tavolo cliccato in precedenza, di impostare nei pannelli successivi l'ordine relativo a quest'ultimo.
Durante l'inizializzazione del pannello vengono creati anche le quattro schermate relative ai medesimi pulsanti. Queste schermate sono :"pizzaOrderingJPanel", "beveraOrderingJPanel" ,"focacciaOrderingJPanel", "OrderViewer". Nell'ultimo caso viene creato un oggetto di tipo "OrderViewer", che si occupa di permettere all'utente di modificare l'ordine considerato. Mentre nei primi tre casi, vengono creati dei pannelli di tipo "OrderingJPanel".

Questo tramite l'oggetto "myOrderingPanelCreator", che facilita la creazione del pannello per l'ordinazione da creare. A tale scopo è necessario impostare vari parametri.
Infatti un oggetto di tipo "OrderingJPanel" necessita di tre array di stringhe: il primo specifica il nome dei casi base del decorator; il secondo il nome dei “toppings”; il terzo il nome di tutte le classi prese in considerazione all'interno del pannello. Inoltre deve essere specificato il percorso dell'immagine di sfondo e la tipologia di decorator che si sta attualmente trattando("pizza","focaccia" o "beverage").

A questo punto, cliccando sul pulsante "pizza", verrà reso visibile il pannello di ordinazione relativo alle pizze. All'interno del quale è possibile, cliccando sugli appositi pulsanti , selezionare la pizza che si desidera. Il pannello si presenta come l'insieme dei tre casi base e dei “toppings”(relativi, ovviamente, al cibo scelto). Ognuno di essi è un pulsante "JButtonTextImage", al quale viene modificato il colore del bordo una volta selezionato. Nel momento in cui si è scelto la propria pizza e si preme su tasto "Add To Order", saranno creati gli oggetti delle classi scelte. Quest'ultimo passaggio è reso possibile dall'oggetto myItemCreator, che attraverso specifici metodi, crea il caso base del decorator selezionato, per poi arricchirlo con i “toppings” scelti in precedenza.

Una volta ordinato un piatto è possibile modificare o semplicemente, visionare, il proprio ordine. Questo attraverso il pannello "OrderViewer", raggiungibile tramite il pulsante "Order" nella schermata "OrderManger". All'interno del pannello, sono presenti tre pulsanti (fatta eccezione per il tasto "Back"): "Edit" utilizzato insieme a "Remove" per rimuovere un piatto dal proprio ordine; "Pay Order" che rimanda alla schermata di pagamento. Inoltre è presente un oggetto "scrollPane" di tipo "OrderScrollPane", che serve a visualizzare tutti i dettagli dell'ordine corrente, man mano che viene aggiornato. Anche in questa classe, come nelle precedenti, i pulsanti e le etichette vengono creati tramite un oggetto "myComponentCreator" di tipo “ComponentCreator”, tale oggetto viene utilizzato, dopo aver impostato il componente che si intende creare e formattare, attraverso il metodo apposito.
Viene inoltre utilizzato l'oggetto "myHtmlFormatter", che contiene varie formattazioni per il testo in html. Tali formattazioni sono utilizzate per modificare l'etichetta nella quale appaiono i dettagli dell'ordine corrente.

Una volta cliccato sul pulsante relativo al pagamento, viene visualizzato il pannello dove sarà possibile scegliere il metodo desiderato. Al suo interno sono presenti tre pulsanti, anch'essi istanze della classe "JbuttonTextImage". Ognuno dei quali, una volta premuto, crea un oggetto adeguato, per eseguire il pagamento tramite il metodo corrispondente. Il tutto utilizza il design pattern Strategy, i tre metodi di pagamento infatti, implementano l'interfaccia "PayMethodInterface", e conseguentemente il suo metodo "payMethod". A seconda del quale ogni classe ("CashPayment", "BancomatPayMethod", "MobilePayMethod") aggiunge determinati componenti grafici ad un pannello comune,al fine di consentire l'operazione. Portata a termine quest'ultima, tutti gli elementi(classi che estendono l'interfaccia "Item") dell'ordine corrente vengono cancellati, permettendo cosi' all'utente di poter eseguire nuove ordinazioni.

Descrizione delle classi che compongono il progetto(divise per pacchetti di appartenenza).

pacchetto pizzeriadiddieffe.core

1- **CreateObjectByName**

Classe che implementa l'interfaccia "ItemsCreatorInterface", viene utilizzata per creare i vari oggetti che andranno a comporre un singolo item. Tale elemento puo' essere di tipo pizza, beverage o focaccia.
E' composta da due metodi principali: "createObjectByName", in grado di creare tramite le informazioni contenute nei parametri un oggetto che rappresenta un "caso base" del pattern Decorator; "createToppingByName", la cui funzione e' praticamente la stessa di "createObjectByName" , ad eccezione però, che in questo caso vengono istanziati e restituiti degli oggetti rappresentanti i topping del pattern Decorator.

2- **Item**

Interfaccia che viene implementata da diverse classi del programma, specifica i vari metodi che dovranno essere implementati nelle sottoclassi. Quali "getInfo", getPrice", "add", "remove".

3- **ItemCreatorInterface**

Interfaccia che viene implementata dalla classe "CreateObjectByName", specifica i metodi principali che la classe dovra' implementare. Quali "createObjectByName" e "createToppingByName".

4- **Order**

Classe che implementa l'interfaccia "Item", implementa quindi tutti i metodi descritti in quest'ultima. Al suo interno dichiara inoltre vari campi: "id", che viene utilizzato per differenziare i vari ordini uno dall'altro; "myOrder", una LinkedList utilizzata per immagazzinare tutti gli item che andranno a comporre l'ordine considerato. La classe specifica inoltre dei metodi getters e setters per il campo "id", oltre che a due metodi "deleteAll" e "deleteIndex", che rispettivamente, eliminano dalla LinkedList "myOrder" tutti gli item contenuti, oppure l'item che si trova in una certa posizione (che sara' inserita come parametro del metodo).

5- **OrderManager**

Classe che implementa l'interfaccia "OrderManagerInterface". Utilizza una LinkedList "myOrderList" per gestire i vari ordini che vengono creati a runtime. E' in grado di cercare un ordine all'interno della propria lista o di crearlo in caso di fallimento, tramite il metodo "createOrder".

6-**Sound**

Classe che implementa l'interfaccia "SoundPlayerInterface". Viene utilizzata per creare e riprodurre suoni relativi alla pressione di pulsanti. Dichiara quindi un campo "soundName", che puo' essere impostato piu' volte tramite il metodo "setSound".

7- **SoundPlayerInterface**

Interfaccia implementata dalla classe "Sound", dichiara due metodi:
"setSound", utilizzato per impostare il nome del file audio che verra' riprodotto;
"playSound", metodo utilizzato per riprodurre il file precedentemente impostato.

pizzeriadiddieffe.core.pizza/beverage/focaccia

1- **Pizza**

Classe astratta che viene estesa dalle classi "PizzaDecorator" e da tutti i casi base del pattern ("BasicBrownPizza", "BasicDoubleWhitePizza", "BasicWhitePizza").
Implementa l'interfaccia "Item" e dichiara due metodi astratti "getInfo" e "getPrice".

2- **PizzaDecorator**

Classe astratta che estende la classe "Pizza", dichiara due metodi astratti "getPrice" e "getInfo",che dovranno quindi essere implementati dalle sottoclassi, inoltre implementa i metodi "add" e "remove" dell'interfaccia "Item".

3- **BasicWhitePizza**

Classe che estende la classe astratta "Pizza", implementando i metodi "getPrice" e "getInfo". Utilizza due campi “PRICE” e “INFO”, che specificano rispettivamente il prezzo dell'oggetto corrente e la sua descrizione.
Vengono inoltre implementati i metodi "add" e "remove" dell'interfaccia "Item".
Viene considerato come il caso base del pattern Decorator, potra' quindi essere decorato dai vari topping presenti nel pacchetto ".topping" .

pizzadiddieffe.core.pizza.topping

1- **Anchovy**

Classe che estende “PizzaDecorator”, fa uso dei metodi "getInfo" e "getPrice" per decorare l'oggetto tempPizza. Quest'ultimo viene passato come parametro al costruttore della classe. Viene utilizzato come topping del pattern Decorator, e' quindi sufficiente impostare come "tempPizza" una qualunque classe che estende la classe astratta "Pizza", al fine di arricchirla con le informazioni della classe “Anchovy”.

pizzeriadiddieffe.gui

1- **OrderViewer**

Classe che implementa l'interfaccia OrderViewerInterface, della quale implementa l'unico metodo "setOrder", utilizzato per far riferimento all'interno della classe ad un ordine specifico. Inoltre estende la classe "JpanelWithBackgroundImgAndBackButton", della quale eredita i pulsante per navigare fra i pannelli e vari metodi utilizzati per la realizzazione dell'interfaccia. All'interno della classe si fa uso di uno scrollpanel, utilizzato per visualizzare tutte le informazioni relative all'ordine corrente. Sono inoltre presenti quattro pulsanti: "back" che viene utilizzato per tornare alla schermata precedente; "Edit", utilizzato per selezionare i vari elementi ordinati in precedenza; "Remove", utile per rimuovere dall'ordine uno o piu' oggetti indesiderati; "Pay Order", pulsante utilizzato per accedere alla schermata di pagamento.
Dichiara inoltre un oggetto "HtmlFormatter", utilizzato per formattare il testo dell'ordine considerato in html.

2- **OrderViewerInterface**

Interfaccia che dichiara un solo metodo, "setOrder",utilizzato nelle classi che la implementano per impostare l'ordine sul quale intendono lavorare.

3- **PayChooser**

Classe utilizzata per impostare la strategia di pagamento designata dal cliente, nel pannello di scelta dei pagamenti "PayOrderJPanel". Utilizza il costruttore a tale scopo, mentre dichiara il metodo "createPanel", che a sua volta richiama il metodo "payMethod" della strategia impostata, per effettuare il pagamento.

4- **PizzeriaDiddieffeUI**

Classe grafica principale, utilizza vari oggetti e metodi per creare i pannelli che saranno mostrati nel corso applicazione. Istanzia ed utilizza: "myComponentCreator", utilizzato per creare i vari pulsanti e le etichette, tramite le classi specializzate "FormattedLabel" e "FormattedButton"; vari pulsanti ai quali spetta la funzione di far impostare all'utente il numero di tavoli presenti all'interno ed esterno del locale; vengono inoltre creati quattro pannelli, complessivamente, nei metodi "initialize" e "createPlacesPanels". Nel primo si crea il pannello che permettera' la scelta interno/esterno, nel secondo si creano invece i pannelli che conterranno i vari tavoli: "Inside" per i tavoli interni ed "Outside" per quelli esterni. Infine viene creato il pannello che permettera' di scegliere se ordinare dei cibi o pagare il conto relativo all'ordine del tavolo.

5- **Table**

Classe utilizzata nei pannelli interno/esterno, rappresenta il tavolo che si intende selezionare. Estende la classe "ClickableButtonWithImage", ed e' in sostanza un “JButton” con l'immagine preimpostata che raffigura un tavolo apparecchiato.

pizzeridaddiddeffe.gui.componentclasses

1- **ComponentFormatterInterface**

Interfaccia che viene estesa dalle classi che intendono formattare un componente grafico, dichiara i metodi: "setComponentProp" utilizzato per impostare le caratteristiche sul pannello del componente ("Bounds"); "setComponentTextProp" utilizzato per impostare le caratteristiche inerenti al testo del componente; "createNewComponent" viene usato per creare un nuovo oggetto del componente che si vuole "formattare"; "getFormattedComponent" restituisce il componente formattato in precedenza.

2- **ComponentGetterInterface**

Interfaccia implementa dalla classe "ComponentGetter". Dichiara un unico metodo, che deve restituire una lista di componenti presenti all'interno di un "container" specificato come parametro.

3- **ComponentSetter**

Classe che utilizza un campo "ComponentFormatterInterface" per impostare una classe da utilizzare in seguito,a patto che quest'ultima implementi la suddetta interfaccia. Vengono inoltre utilizzati dei metodi per creare e formattare il nuovo componente, sempre in base alla classe scelta. Il metodo "setCurrentComponent" e' responsabile della creazione del nuovo componente,che in seguito verra' formattato tramite i metodi "setComponentTextProp" e "setComponentProp", che ne specificano, rispettivamente, le proprieta' del testo e le dimensioni sul pannello. Infine tale componente viene restituito tramite il metodo "getFormattedComponent", che come gli altri, non fa altro che richiamare il metodo corrispondente nella classe definita precedentemente.

4- **ComponentGetter**

Classe che implementa l'interfaccia ComponenteGetterInterface, implementa quindi il metodo "getCompoenents", che viene richiamato finche' non restituisce una lista di componenti.

pizzeriadiddieffe.gui.creators

1- **ComponentCreator**

Classe che implementa l'interfaccia "ComponentCreatorInterface", utilizza tre oggetti: "myButtonFormatter" in grado di "formattare" le proprieta' del testo e le sue coordinate; "myLabelFormatter" utilizzato invece, per formattare una "label"; infine "myComboBoxFormatter", che viene utilizzato per modificare le proprieta' di una "Combobox". La classe si avvale di un oggetto "ComponentFormatter" per applicare le modifiche al componente interessato, richiamando i metodi designati. Infine restituisce il componente scelto tramite i metodi "getLabel", "getButton", "getComboBox".

2-**ComponenteCreatorInterface**

Interfaccia implementata dalla classe ComponentCreator, dichiara i metodi: "createButton", "createLabel", "createComboBox", utilizzati per creare i componenti corrispondenti; "setUpComponentProp" per definirne le caratteristiche; "getButton", "getLabel", "getComboBox" per restituire il componente formattato in precedenza.

3- **ItemCreator**

Classe che implementa l'interfaccia ItemCreatorInterface. Vengono quindi implementati i metodi: "setItemLists" utilizzato per impostare le liste che contengono, rispettivamente, il nome dei toppings, i nomi dei casi base e i nomi delle classi da creare; "setCurrentItem" viene utilizzato per impostare un oggetto di tipo "Item", necessario alla creazione dei toppings; "createChoosenBaseCase" che si occupa di creare, tramite le liste precedentemente impostate e il nome della classe "className", il caso base scelto dall'utente; "getCurrentItem", resituisce l'oggetto appena creato; "createChoosenToppings" utilizzato per creare il topping scelto e decorare con esso il caso base presente nel campo "currentItem". Nel processo vengono usate delle liste contenti le informazioni relative al nome delle classi, al nome dei toppings, al testo utilizzato nei vari pulsanti per distinguere i “toppings” e i casi base. I vari oggetti vengono quindi creati tramite l'oggetto myCreator, istanza della classe “CreateObjectByName”.

4- **ItemCreatorInterface**

Interfaccia implementata dalla classe "ItemCreator", dichiara i metodi "setItemsList", necessario per impostare le corrette liste utilizzabili nella fase di creazione degli oggetti; "setCurrentItem" utilizzato per impostare l'oggetto che si intende decorare; "createChoosenBaseCase" metodo che crea il caso base che ha per nome il testo del parametro "className"; "creatChoosenToppings" metodo responsabile della creazione dei “toppings”; "getCurrentItem" restituisce l'oggetto creato, eventualmente decorato con i vari “toppings”.

5- **OrderingPanelCreator**

Classe che implementa l'interfaccia "PanelCreator", viene utilizzata per creare dei pannelli "OrderingPanel". Definisce i metodi dell'interfaccia e li implementa per impostare tutti i campi e array necessari,al fine di poter creare l'oggetto "OrderingPanel" stabilito. Utilizza il metodo "setParameters" per impostare le proprieta' del pannello e la sua immagine di sfondo;il metodo "createPanel" al fine di creare il pannello e impostare i vari array di cui quest'ultimo necessita; viene impostato il pannello precedente come ritorno per il tasto “back” del pannello "Ordering"; infine utilizza il metodo “getOrderingPanel” per restituire il pannello appena creato e "formattato".

6- **PanelCreator**

Interfaccia implementata nella classe "OrderingPanelCreator". Dichiara i metodi: "setParameters" utilizzato per impostare i vari parametri necessari alla creazione del pannello "Ordering" corrente; "createPanel" utile per impostare i vari array di cui necessita la classe "OrderingPanel" per creare, posizionare e modificare i vari elementi al suo interno (JButtons,casi base e toppings); "setOrderingBackButton" utilizzato per impostare il pannello al quale il pulsante "back" del pannello ordering si riferisce; "getOrderingPanel" restituisce il pannello appena creato.

pizzeriadiddieffe.gui.jpanel

1- **JPanelWithBackBtn**

La classe estende "JPanel", arricchendo il pannello con il pulsante “Back” in alto a sinistra. Questo viene implementato tramite il metodo “paintBackBtn” , che aggiunge il pulsante menzionato. Il metodo “setPanelVisibility” fa in modo che una volta premuto il pulsante “Back” esso imposti un determinato pannello visibile, mentre il resto del frame viene impostato come invisibile.

2- **OrderingJPanel**

La classe estende "JPanelWithBackgroundImgAndBackBtn" quindi oltre ad essere un pannello con sfondo e pulsante “Back” esso crea tutti i componenti per comporre l’ordine.
Riceve in ingresso le stringhe con i componenti da visualizzare (nomi casi base, nomi "toppings", e relativi nomi delle classi).
Il metodo “createBaseItems” crea i pulsanti dei casi base, aggiungendo all’ActionListener anche la possibilità di cambiare colore del bordo di ogni pulsante. In modo che, una volta selezionato, il colore del bordo sara' diverso rispetto a quello di base.
Il metodo “createToppingsItems” fa la stessa cosa del metodo appena descritto, ma per i "toppings". Essi verranno resi selezionabili, solo se e' stato precedentemente selezionato un caso base.
Infine e' presente il metodo “addOrderButton”. Quest'ultimo permette di visualizzare il pulsante per completare l’ordine ed emettere un suono a ordine completato.
Aggiunge anche un pulsante “Clear” che serve per deselezionare tutti i componenti selezionati in precedenza, richiamando il metodo “resetSelection”.

3- **OrderScrollPane**

La classe estende "JScrollPane" che non è altro che un pannello al quale viene aggiunta possibilità di essere scorso, nel caso in cui non ci fosse abbastanza spazio per contenere tutti i componenti. Viene usato nell’"OrderViewer" permettendo quindi all'utente di poter visualizzare tutti i dettagli dell'ordine.

4- **PayOrderJPanel**

Estende "JPanelWithBackgroundImgAndBackBtn" inserendo all’interno del pannello designato, la scelta per il metodo di pagamento ("Cash"," Bancomat", "Mobile").
Il metodo “PayOrderJPanel” imposta le dimensioni e la posizione dei pulsanti, che vengono creati in seguito,tramite il metodo “createPayButton”. Infine “createPayPanel” crea il pannello in base al tipo di pagamento selezionato, infatti tramite il metodo “getStrategy”, viene restituita ed impostata una classe che implementa l'interfaccia "PayMethodInterface". Quindi si fa uso del metodo "payMethod", per aggiungere i componenti adeguati al pannello.

5- **ScrollPaneInterface**

Interfaccia che viene implementata dalla classe "OrderScrollPane". Vengono dichiarati inoltre i metodi: "setCurrentJPanel",necessario per impostare il pannello su cui si intende lavorare; "setParameters" utilizzato per impostare i "bounds" del pannello; "setColors", utile per impostare le proprieta' grafiche(colore del bordo, colore di sfondo) dell'oggetto; "getScrollPanel" che restituisce il pannello appena modificato.

pizzeriadiddieffe.gui.jpanel.japanelwithbackground

1- **JPanelWithBackgroundImgAndBackBtn**

Classe che estende "JPanel” arricchendo il pannello con un immagine e con un pulsante “Back”, posizionato in alto a sinistra. L’immagine di sfondo viene passata tramite una String che ne indica l’indirizzo. La classe stessa cerca e crea l’immagine come "Image", ad essa vengono impostate le dimensioni desiderate, per poi utilizzarla come del pannello. Inoltre viene sovrascritto il metodo “paintComponent”, in modo da disegnare l’immagine appena creata.
Il metodo “paintBackButton” aggiunge il pulsante “Back” in alto a sinistra di ogni pannello creando un “ClickableButtonWithImage”. Quando esso viene premuto viene reso visibile il pannello precedente,impostato precedentemente tramite il metodo “setPanelVisibility”.
Quest’ultimo e' infatti in grado di impostare, tramite l'oggetto di istanza della classe "ComponentsGetter", uno specifico pannello come "ritorno"(visibile) del tasto "Back".

2- **JPanelWithBackgroundOrder**

La classe estende “JPanelWithBackgroundImgAndBackBtn” inserendo nel pannello designato, quattro pulsanti di scelta per l’utente: "Beverage", "Pizza", "Focaccia", "Order". L'inserimento avviene tramite il metodo “drawChoiceButtons”. Inoltre, alla classe vengono passate tutte le informazioni necessarie, per creare i pannelli successivi, che permetteranno di utilizzare gli ordini.
Metodo “createOrderViewerPanel” utilizzato per creare il pannello relativo al tasto "Order". Viene quindi creato un oggetto “OrderViewer”,che impostera' come visibile il pannello che permette la scelta delle varie operazioni, relative al tavolo scelto.
Il metodo “createOrderingPanels” viene invocato per creare i tre pannelli di ordinazione dei cibi, relativi ai pulsanti("Pizza", "Focaccia", "Beverage"). Viene quindi creato il pannello con tutte le scelte per ordinare in base a cosa viene passato come parametro del metodo, sotto forma di stringa. Queste stringhe conteranno la lista di tutte le componenti base e topping dell’articolo scelto, compresi indirizzi delle immagini e nome delle classi ai quali appartengono.
Il “createOrderingPanel” crea il singolo pannello passando tutti i parametri stringhe menzionati sopra. Infine, “setTableId” imposta l’"id" del tavolo per cercare poter lavorare sull’ordine desiderato.

3- **JPanelWithBackgroundTables**

Classe che estende “JpanelWithBackgroundImgAndBackBtn”, viene utilizzata per visualizzare il numero dei tavoli scelti all’inizio dell’applicazione. Questi sono posizionati all’interno del pannello tramite il metodo “drawTables”. Ogni tavolo sara' rappresentato da un pulsante (oggetto di tipo "Table") che renderà visibile, una volta cliccato, un altro pannello passandogli l’"Id" del tavolo.
Il metodo “getCoordinates” è stato implementato per rendere la grandezza e la posizione dei tavoli più intelligente, in base al numero dei tavoli scelti inizialmente.

4- **JPanelWithImageOrderInterface**

Interfaccia contente il metodo “setTableId” che andrà ad impostare l’"Id" per il tavolo corrente. Viene implementata in tutte le classi del pacchetto,che estendono "JPanel" e necessitano di un oggetto "Order", per portare a termine le varie operazioni.

5- **JPanelWithImageInterface**

Interfaccia implementata dalla classe "JpanelWithBackgroundImage", quest'ultima ne implementera' in metodi "setVisiblePanel" e "getFrame".

pizzeriadiddieffe.gui.jbutton

1- **ClickableButtonWithImage**

La classe estende "JButton", aggiungendo come sfondo un immagine. Questa viene impostata tramite un parametro del costruttore, insieme alle dimensioni e alla posizione del pulsante. Quest'ultimo ha la proprietà di cambiare immagine di sfondo, quando il mouse lo clicca o si trova al di sopra di esso.

2- **JButtonTextImage**

Estende “ClickableButtonWithImage” ed imposta il colore del testo del pulsante, in scegliendolo in base a quest'ultimo, in modo da renderlo più leggibile.

pizzeriadiddieffe.gui.formattedelements

1- **ButtonBorderManager**

Classe che implementa”ButtonBorderManagerInterface”.
Il metodo “getBorderColor(JButtonTextImage currentItemButton)” restituisce il colore del bordo del JButtonTextImage passato al metodo.
Il metedo “changeBorderColor(JButtonTextImage currentItemButton)” assegna al bordo del JButtonTextImage passato al metodo il colore desiderato.
Il metodo “resetBorderColor(JButton currentButton)” resetta il colore del bordo del JButton passato al metodo al colore di default.
Il metodo “setOthersButtons(boolean enable, LinkedList<JButton> list, JButton myButton)” data una lista di JButton, un singolo JButton e un valore booleano, agisce sui JButton della lista attivandoli.
Il metodo “getButtonListIterator(LinkedList<JButton> list)” restituisce l'iteratore della lista passata al metodo.
Il metodo “resetButtons(LinkedList<JButton> baseCasesButtonsList, LinkedList<JButton> toppingButtonList)” resetta le liste di JButton passate al metodo.
Il metodo “resetListButtons(Iterator<JButton> iterator)” resetta il colore del bordo dei JButton della lista passata al metodo.

2- **ButtonBorderManagerInterface**

Interfaccia contenete i metodi “getBorderColor(JButtonTextImage currentItemButton)”, che restituisce il colore del bordo del JButtonTextImage passato al metodo, “changeBorderColor(JButtonTextImage currentItemButton)”, che setta il colore del bordo del JButtonTextImage passato, “setOthersButtons(boolean enable, LinkedList<JButton> list, JButton myButton)”, che data una lista di JButton, un singolo JButton e un valore booleano, agisce sui JButton della lista attivandoli, e “resetButtons(LinkedList<JButton> baseCasesButtonList, LinkedList <JButton> toppingButtonList)”, che resetta le liste di JButton passate al metodo.

3- **FormattedButton**

Classe che implementa “ComponentFormatterInterface”.
Il metodo “setComponentProp(int x, int y, int width, int height)” setta le impostazioni del JButton assegnato alla variabile nella classe.
Il metodo “setComponentTextProp(String text, String font, int fontSize, Color textColor)” setta le impostazioni del testo del JButton assegnato alla variabile nella classe.
Il metodo “createNewComponent()” crea un JButton e lo assegna alla variabile della classe.
Il metodo “getFormattedComponent()” restituisce il JButton creato nella classe.

4- **FormattedComboBox**

Classe che implementa “ComponentFormatterInterface”.
Il metodo “setComponentProp(int x, int y, int width, int height)” setta le impostazioni del JComboBox assegnato alla variabile nella classe.
Il metodo “setComponentTextProp(String text, String font, int fontSize, Color textColor)” setta le impostazioni del testo del JComboBox assegnato alla variabile nella classe.
Il metodo “createNewComponent()” crea un JComboBox e lo assegna alla variabile della classe.
Il metodo “getFormattedComponent()” restituisce il JComboBox creato nella classe.

5- **FormattedLabel**

Classe che implementa “ComponentFormatterInterface”.
Il metodo “setComponentProp(int x, int y, int width, int height)” setta le impostazioni del JLabel assegnato alla variabile nella classe.
Il metodo “setComponentTextProp(String text, String font, int fontSize, Color textColor)” setta le impostazioni del testo del JLabel assegnato alla variabile nella classe.
Il metodo “createNewComponent()” crea un JLabel e lo assegna alla variabile della classe.
Il metodo “getFormattedComponent()” restituisce il JLabel creato nella classe.

6- **HtmlFormatter**

Classe che implementa “HtmlFormatterInterface” ed ogni metodo restituisce l'elemento di formattazione in HTML assegnati alle variabili corrispondenti.

7- **HtmlFormatterInterface**

Interfaccia contenente i metodi “getBullet()”, “getComma()”, “getTabSpace()”, “getEndItalic()”, “getNewLine()”, “getStartBold()”, “getEndBold()”, “getPrice()”, “getHighligh()” e “getEndhighligh()”, che restituiscono elementi di formattazione del testo in HTML.

pizzeriadiddieffe.gui.paymethods

1- **BancomatPayMethod**

Classe che estende “JpanelWithBackBtn” ed implementa “payMethodInterface”.
Il metodo “createTotalLabel()” crea una JLabel, nella quale c'e' una stringa di testo predefinita “Amount to Pay” piu' l'importo totale dell'ordine.
Inoltre il metodo crea un JButton “Pay” al quale viene aggiunto un ActionListener per cui, se il codice inserito dall'utente e' minore di 5 cifre alla pressione il testo del JButton viene settato a “Insert Code”, altrimenti viene settato a “Payed!” e l'ordine viene svuotato.
Il metodo “createButtons()” crea invece una JLabel che ha come stringa “Insert Code ”, crea poi 10 JButton(numeri da 0 a 10), che vengono disposti richiamando il metodo “setCordinates(int i)”, ai quali viene aggiunto un ActionListener per cui alla pressione di uno di questi JButton viene inserito un asterisco nella JLabel.
Viene infine creato un JButton “C” che, se premuto, azzera il codice inserito.

2- **CashPayment**

Classe che estende “JpanelWithBackBtn” ed implementa “payMethodInterface”.
Il metodo “createTotalLabel()” crea una JLabel, nella quale c'e' una stringa di testo predefinita “Amount to Pay” piu' l'importo totale dell'ordine.
Inoltre il metodo crea un JButton “Pay” al quale viene aggiunto un ActionListener per cui, se quando viene premuto il resto(cioe' la differenza tra l'importo totale dell'ordine e quello inserito dall'utente) e' maggiore di 0 o il testo del JButton e' uguale a “Pay” setta il testo del JButton a “Change: ” piu' il resto, altrimenti setta il test del JButton a “Payed!” e svuota l'ordine.
Il metodo “createButtons()” crea invece una JLabel che ha come stringa “Amount Paid ”, crea poi 10 JButton(numeri da 0 a 10), che vengono disposti richiamando il metodo “setCordinates(int i)”, ai quali viene aggiunto un ActionListener per cui alla pressione di uno di questi JButton viene inserito nella JLabel l'intero corrispondente al testo del JButton premuto ricavato attraverso il “parseInt(String text)”.
Vengono infine creati due JButton “C” e “.” che, se premuti, rispettivamente azzerano la parte di JLabel inserita dall'utente e inseriscono un “.” nella JLabel.

3- **MobilePayMethod**

Classe che estende “JpanelWithBackBtn” ed implementa “payMethodInterface”.
Il metodo “createBox()” crea due JComboBox inizialmente vuote, alla prima vengono aggiunte le scelte “Contact Less” e “Code Insertion” richiamando il metodo “createMenuItems(JComboBox<String> currentMenu, String[] itemsName)”, alla seconda vengono invece aggiunte le scelte "Apple Pay", "Android Pay" e "Generic Operator" richiamando il metodo “createMenuItems”.
Il metodo “createTotalLabel()” crea una JLabel, nella quale c'e' una stringa di testo predefinita “Amount to Pay” piu' l'importo totale dell'ordine.
Crea poi altre due JLabel: la prima, “Choose Autentication Method”, posizionata sopra la prima JComboBox; la seconda, “Choose Operator”, posiziona sopra la seconda JComboBox.
Infine il metodo crea un JButton “Pay” al quale viene aggiunto un ActionListener per cui, se quando viene premuto il testo del JButton e' uguale a ”Pay” setta il testo del JButton a “Autentication” e disabilita le due JComboBox, mentre se il testo del JButton e' uguale ad “Autentication” setta il test del JButton a “Payed!”, disabilita il JButton e svuota l'ordine.

4- **PayMethodInterface**

Interfaccia contenente il metodo “payMethod(Order totPrice, JPanelWithBackgroundImgAndBackBtn myPanel)”, che setta l'ordine ed il pannello correnti.

Descrizione dei tests

pizzeriadiddieffe.test

1- **BeerTest**

Classe di test che controlla il corretto funzionamento delle classi di tipo topping. A tale scopo crea un oggetto fittizio del tipo "EmptyLargeGlass"(caso base del decorator), successivamente utilizza quest'ultimo per effettuare dei controlli sulla descrizione e sul prezzo precedenti e successivi alla decorazione effettuata dalla classe "Beer".

2- **ComponentSetterButtonBoundsTest**

Nella classe di test vengono controllati i metodi della classe "ComponentSetter" necessari per impostare i "bounds" del componente designato. Si fa uso,conseguentemente, dell'interfaccia "ComponentFormatterInterface" e della classe "FormattedButton",utilizzata come esempio chiave. Vengono quindi testati i metodi delle rispettive classi, affinche' vengano attribuiti al pulsante designato, le corrette coordinate e dimensioni("Bounds").A tale scopo vengono impostate le varie propieta' del pulsante, per poi verificarne la correttezza.

3- **ComponentSetterButtonTextPropTest**

Altra classe di test incaricata di controllare,come la precedente, il funzionamento della classe "FormattedButton" e quindi, dell'interfaccia "ComponentFormatterInterface".In questo caso vengono pero' testati i metodi responsabili della formattazione del testo relativa al pulsante scelto. Anche in questa classe, vengono quindi utilizzati i metodi della classe per: impostare le proprieta' desiderate; controllare se sono state applicate in modo corretto.

4- **CreateCaseByNameTest**

Classe di test utilizzata per controllare le funzioni della classe "CreateObjectByName".Si verifica quindi che i metodi eseguiti,dalla classe, per istanziare nuovi oggetti di tipo "caso base" e "topping" producano, effettivamente gli oggetti voluti. Vengono quindi testati i metodi necessari per la creazione degli oggetti appena menzionati. In seguito ne vengono controllate le informazioni, al fine di poterne assicurare la correttezza.

5- **EmptyLargeGlassTest**

In questa classe di test viene testato il caso base del decorator, "EmptyLargeGlass". Nel fare cio', si fa uso di un altro oggetto, definito come "fakeItem",esso viene quindi utilizzato,per controllare che a seconda del metodo invocato, l'oggetto venga inserito o meno all'interno del caso base "EmptyLargeGlass".

6- **GetComponentsTest**

Classe che controlla le funzionalita' della classe "ComponentsGetter".Viene quindi testato il metodo utilizzato per restituire un determinato componente,nello specifico, si testa la possibilita' di restituire un pulsante o un'etichetta dal frame all'interno del quale sono stati inseriti.

7- **ItemTestClass**

Questa non e' una vera e propria classe di test, ma un oggetto creato per poter testare le funzionalita' dei diversi casi del pattern "Decorator".

8- **OrderManagerTest**

Vengono testate le funzionalita' della classe "OrderManager", quali: la possibilita' di restituire un oggetto,precedentemente creato, della classe "Order"; la possibilita' di creare un nuovo oggetto della suddetta classe, qualora il medesimo non sia gia' presente all'interno della lista di ordini creati.

9- **OrderTest**

Si testano tutti i metodi della classe "Order", attraverso un oggetto di tipo "ItemTestClass", che viene inserito e rimosso dall'ordine piu' volte. Ognuna delle quali viene verificato che le informazioni,quali prezzo e descrizione degli oggetti presenti all'interno dell'ordine, siano corrette.

10- **SoundTest**

Classe che testa la correttezza della classe "Sound", in grado di riprodurre un file audio impostato tramite un apposito metodo. Viene fatto uso quindi di piu' percorsi a file audio(presenti o meno all'interno della cartella relativa), che si tenta di riprodurre tramite l'oggetto di tipo "Sound".