Progetto OOP

GRUPPO 18

Burger chose

L’applicativo si prefigge l’obbiettivo di fornire un’interfaccia che permette all’utente/cliente di comporre un’ordinazione che può contenere più panini personalizzati a suo piacimento attraverso un menù “a spunta”. Per identificare in modo univoco l’ordinazione viene assegnato un numero incrementale al momento della creazione dell’ordine, inoltre viene richiesto all’utente di inserire anche un cognome indicativo che verrà usato al momento della consegna (per interfacciarsi in maniera più semplice con il cliente, se questo non inserisce un cognome attendibile, come ad es. un numero, non è un problema essendo l'ordinazione identificato solamente in base al numero univoco lasciato al cliente). Il programma provvede a lanciare un’eccezione nel caso in cui un ordinazione non presenta la scelta di un panino. Una volta confermata l’ordinazione, questa viene inserita in una coda delle ordinazioni (gestita appunto secondo una politica FIFO), la quale viene utilizzata dai cuochi della paninoteca per prelevare le ordinazioni e preparare i rispettivi panini. Quando un cuoco termina un ordine, provvede a salvare in una struttura il numero dell’ordinazione, il cognome e il conto di quest’ultimo. Infine il cassiere accederà a questa struttura nel momento in cui un cliente deve saldare il conto, per poi eliminare dalla struttura il riferimento a quel cliente.

Altre funzionalità:

E' previsto un backup su file di testo del buffer delle ordinazioni prese e delle ordinazioni servite ogni volta che il contenuto dei rispettivi buffer cambia; in caso di anomalia il programma può essere riavviato alle condizioni di arresto).

Il programma inoltre prevede di validare o no dei possibili sconti in base ai numeri di panini ordinati; in particolare, se l'ordine:

> supera i 3 panini, si ha uno sconto del 15% sul conto complessivo

> supera i 9 panini, si ha uno sconto del 20% sul conto complessivo

COME TESTARE IL PROGRAMMA

Creare un nuovo progetto in NetBeans, estrarre il file zip e copiare la cartella “OOP2017” in “src”, questa cartella rappresenta il package e quindi quello generato di default alla creazione del progetto deve essere cancellato.

Il file di test da utilizzare è *Test* e si trova nel package “*burger.choice”.*

Lanciando questo file, vengono chiesti dei parametri per settare l'avviamento del locale (sono stati inseriti dei controlli se il numero dei cuochi inseriti non è coerente, ossia se non è un numero oppure se è un numero minore di 0). Successivamente si apre un'interfaccia grafica, che rappresenta il terminale attraverso il quale l'utente potrà inserire le proprie ordinazioni.

Le interfacce in realtà sono 4, ma UI è quella principale, le altre sono create e chiamate da questa.

L'utente tramite questa può inserire il proprio cognome per iniziare a compilare la propria ordinazione componendo i panini e aggiungendoli a quest'ultima. Cliccando sul bottone “Fine” si completa l'ordinazione, tramite il bottone “Aggiungi prossimo panino” è possibile comporre un altro panino da aggiungere alla propria ordinazione e quello composto sarà aggiunto giò all'ordinazione. In qualsiasi momento durante la compilazione dell'ordinazione sarà possibile annullare l'ordinazione. Cliccando sul pulsante “Annulla” infatti, se non è stato ancora aggiunto nemmeno un panino, sarà chiesto all'utente se è sicuro o no di annullare l'ordinazione, se invece ha già aggiunto qualche panino gli

verrà chiesto se vuole annullare l'intera ordinazione, oppure confermarla fino al panino precedente (cioè all'ultimo panino aggiunto, annullando quindi solo la creazione di questo ulteriore panino).

E' possibile monitorare la cronologia degli eventi che accadono attraverso il terminale di NetBeans (es. ordinazione presa, backup effettuato, ecc...).

CLASSI IMPLEMENTATE E LORO DESCRIZIONE

Per rendere eseguibile l'applicazione creare un nuovo progetto in NetBeans e copiare al suo interno la cartella *gruppo18* come package.

Panino

Implementa l'interfaccia *Serializable* e possiede 3 attributi: *descrizione(String), numExtra (int) e prezzo (double).*

Il primo fornisce una descrizione della composizione del panino, utile al cuoco che deve poi comporlo; il secondo indica il numero di ingredienti extra aggiunti al panino oltre quelli previsti: il numero max di extra compresi nel prezzo del panino standard è di 3 (tra formaggi, contorni e salsa); il terzo corrisponde al prezzo del panino (che corrisponde al prezzo del panino standard, senza tener conto degli extra).

Il costruttore della classe prende in ingresso i valori da settare agli attributi, tuttavia se numExtra è maggiore di 0, al valore passato al prezzo del panino, deve aggiungere 0,50€ per ogni extra.

La classe mette a disposizione i seguenti metodi:

getter e setter degli attributi;

String toString() : restituisce la descrizione del panino scelto (con gli extra) e il suo prezzo finale;

Ordinazione

Implementa l'interfaccia *Serializable* e possiede una variabile di classe (cioè statica), che serve per ottenere il numero univoco dell'ordinazione tramite il suo incremento, e altri 4 attributi:

panini (List<Panino>) , rappresenta l'ordinazione attraverso una lista di panini;

numUnivoco (int), indica il numero dell'ordinazione;

cognome(String), il cognome associato all'ordinazione (usato solo per semplicità del cliente);

conto (double), calcolato su tutti i panini dell'ordine e applicando i relativi sconti imposti sulla quantità.

La seguente classe possiede anche un costruttore che prende come ingresso solo il parametro "numUnivoco" , il quale permette di creare un'ordinazione in cui tutti gli attributi sono settati a valori nulli, ad eccetto di numUnivoco, al quale viene assegnato il valore passatogli (Questo per poter poi in modo semplice cercare un'ordinazione sfruttando il metodo equals della seguente classe)

La classe mette a disposizione i metodi di get per gli attributi e di set (ad eccezione del numUnivoco che viene settato solo al momento della creazione dell'ordinazione); inoltre da una propria implementazione ai metodi equals e hashCode sulla base del numero univoco e fornisce:

void aggiungiPanino(Panino panino), che aggiunge un panino alla lista dell'ordinazione di quell'utente;

void calcolaConto(): setta il valore del conto, compreso dello sconto se ci sono le condizioni;

String toString() : restituisce la lista dei panini con la descrizione di ciascun panino dell'ordinazione, seguito dal suo prezzo (usando la toString di panino) e alla fine il prezzo del conto finale;

BufferOrdinazioni

Implementa l'interfaccia *Serializable* e possiede l'attributo "o*rdinazioni (LinkedList<Ordinazione>)",* che rappresenta il buffer delle ordinazioni prese ed è gestita tramite una lista di ordinazioni. Il costruttore viene usato solo se si vuole creare un buffer nuovo (vuoto), e non se si vuole caricare la lista delle ordinazioni dal file di backup. Il file dal quale caricare ha sempre lo stesso nome, ossia "*BufferOrdinazioni.txt*" che è il file di backup appunto.

La classe è thread-safe e autobloccante. Mette a disposizione i seguenti metodi:

void addOrdinazione(Ordinazione ordinazione), permette di aggiungere un'ordinazione in coda;

void removeOrdinazione(int numUnivoco), permette di rimuovere un'ordinazione dal buffer nel caso in cui il cliente decida di annullarla;

Ordinazione consegnaOrdinazione(), permette di prelevare un'ordinazione dalla testa (usata dal cuoco infatti);

void backupOrdinazioni() : salva sul file "bufferOrdinazioni.txt" le ordinazioni nel buffer.

void toString() : restituisce una stringa che contiene la lista di tutte le ordinazioni (una per riga);

Cuoco

Implementa l'interfaccia *Runnable* e possiede 3 attributi: *nome(String), BufferOrdinazioni* e *BufferServiti*, poiché deve prelevare dal primo l'ordinazione da preparare e, una volta completata, questa deve essere inserita nel buffer dei clienti serviti così da poter essere visibile al cassiere quando il cliente si presenterà per pagare il conto.

Il metodo run() è implementato in modo tale che il cuoco si preoccupa di prelevare un'ordinazione dal buffer delle ordinazioni, preparare i rispettivi panini (qui il cuoco aspetta dai 5 ai 10 secondi, per semplicità di funzionamento), e infine aggiungere l'ordinazione completata nel buffer dei clienti serviti.

BufferServiti

Implementa l'interfaccia *Serializable* e possiede l'attributo

"*serviti (HashMap<Integer,Ordinazione>)",* che rappresenta il buffer dei clienti serviti.

Il costruttore viene usato solo se si vuole creare un buffer nuovo (vuoto), e non se si vuole caricare la lista delle ordinazioni servite dal file di backup. Il file dal quale caricare ha sempre lo stesso nome, ossia "*BufferServiti.txt*" che è il file di backup appunto.

La classe è thread-safe e autobloccante. Mette a disposizione i seguenti metodi:

void addServiti(Ordinazione ordinazione), permette di aggiungere un'ordinazione in quelle servite;

Ordinazione consegnaServiti(), permette di prelevare un'ordinazione tra quelle dei clienti serviti in base al numero univoco (usata dal cassiere infatti). Questo metodo lancia l'eccezione *ClienteNonPresenteException* nel caso in cui il numero univoco passato come parametro non identifica nessuna ordinazione nel buffer dei clienti serviti.

void backupServiti() : salva sul file "bufferServiti.txt" le ordinazioni servite nel buffer.

void toString() : restituisce una stringa che contiene la lista di tutte le ordinazioni dei clienti serviti(una per riga);

Su questa classe è stata realizzata la JavaDoc, sulle altre classi solo per i metodi più complessi.

Cassiere

Implementa l'interfaccia *Runnable* e possiede un solo attributo: *BufferServiti*, ossia il buffer dei clienti serviti, così da poter essere visibile al cassiere quando il cliente si presenterà per pagare il conto.

Il metodo run() è implementato in modo tale che il cassiere si preoccupa di prelevare l'ordinazione dal buffer delle ordinazioni servite nel momento in cui il cliente si presenta a saldare. Il cliente deve fornire quindi il numero che identifica l'ordinazione, così il cassiere accede a quell'ordinazione e visualizza il conto che quest'ultimo deve saldare. (Per semplicità di testing si genererà casualmente il presentarsi dei clienti alla cassa, senza chiedere di inserire il numero identificativo da tastiera).

BackupAutomaticoOrdinazioni & **BackupAutomaticoServiti**

Queste due classi si preoccupano, rispettivamente, di salvare su file il buffer delle ordinazioni e il buffer dei clienti serviti. Implementano quindi l'interfaccia Runnable e usano i metodi di backup forniti di buffer stessi (resi thread-safe e autobloccanti).

ClienteNonPresenteException

Viene lanciata dal metodo *consegnaServiti(int numUnivoco)* di BufferServiti, quando il numero univoco inserito non corrisponde ad alcuna ordinazione presente nel suddetto buffer.

OrdinazioneNonPresenteException

Viene lanciata dall'interfaccia quando nel comporre un'ordinazione, non si spunta nemmeno un panino standard.

INTERFACCE

UI (user interface)

Permette di inserire il cognome per iniziare a compilare un'ordinazione. Ha un attributo di tipo *BufferOrdinazioni*, poiché al suo costruttore viene passato il riferimento al buffer già creato nel file di Test (main), al quale devono essere aggiunte le ordinazioni che si completano.

Cliccando sul bottone “avanti”, se non è stato inserito nessun cognome viene mostrato un pannello di errore, in caso contrario viene creata un'ordinazione e lanciata UI2 (passando al suo costruttore il riferimento dell'ordinazione creata e il riferimento del buffer nel quale inserirla una volta completata). Sono stati inseriti in quest'interfaccia un font particolare (importato nel costruttore) e un'immagine, entrambi presenti nel package *Interface*.

UI2

Implementa un menù a spunta che permette all'utente di comporre i panini come preferisce.

Sono stati utilizzati diversi tipi di oggetti:

JButton (Annulla, Fine, Aggiungi prossimo panino);
JRadioButton(che combinati con un ButtonGroup permette di avere la mutua esclusione tra le caselle di selezione nello step 1 della composizione del panino);
JCheckBox (permettono di selezionare contorni, formaggi e salse attraverso una rappresentazione con caselle a spunta);
JComboBox (permettono di selezionare uno tra diversi elementi in essi inseriti (step 2 e 3));

Il bottone “Annulla”, se non è stato ancora aggiunto alcun panino all'ordinazione, apre una finestra di dialogo “Dialog4”, dove chiede all'utente se è sicuro o meno di voler annullare l'ordinazione, mentre se è stato già aggiunto un panino, apre un'altra finestra di dialogo “Dialog3” che chiede all'utente se vuole solo annullare la composizione del panino corrente e confermare l'ordinazione fino al precedente panino inserito, oppure annullare l'intera ordinazione.

Questa classe possiede 2 attributi: ordinazione e bufferOrdinazione, che rappresentano il riferimento all'ordinazione associata al cognome inserito in UI al quale aggiungere i panini che l'utente completa e il buffer in cui andare poi ad inserire quell'ordinazione.

Dialog3 & **Dialog4**

Sono due finestre di dialogo (JDialog) che bloccano il flusso di esecuzione dei JFrame che li chiamano fino a quando l'utente non sceglie una tra le alternative proposte.

OSSERVAZIONE

(* Mettiamo caso che il programma si arresti, data ad esempio un'interruzione di corrente che causa lo spegnimento del dispositivo. Il programma viene fatto ripartire e le ordinazioni ripartono dal numero identificativo “1”. Questo non è un problema poiché la probabilità che l'ordinazione “1” (e anche quelle dei numeri a seguire) è ancora nel buffer delle ordinazioni o dei serviti è pari a 0. Questo caso può presentarsi infatti se il programma viene avviato 2 volte di fila nel giro di pochissimo tempo, ma ciò è improbabile, visto che si suppone anche che un locale che affida la gestione delle

ordinazioni ad un sistema informatico, adotti delle misure di sicurezza in caso di mancanza di corrente (es. presenza di un generatore autonomo di corrente *).

UML