RistoBUS

Gruppo E

Requisiti del cliente e descrizione del dominio del problema

Il famoso ristorante RistoBUS ha richiesto un'applicazione per gestire i menù da proporre ai propri clienti.

Il ristorante propone ai propri clienti una serie di menù prestabiliti. Ogni menù è costituito da un insieme di portate, appartenenti a ben precise categorie (antipasti, primi, secondi, ecc.).

Ogni portata è caratterizzata da codice univoco, descrizione, categoria e prezzo. I clienti possono personalizzare il proprio menù scegliendo, all'atto dell'ordine, una (e una sola) portata fra quelle proposte in ciascuna categoria: il prezzo finale varierà di conseguenza.

I file di testo Portate.txt e Menu.txt contengono rispettivamente l'elenco delle portate e i menù già predisposti, con le possibili varianti

Il RistoBUS vuole che l'applicazione sia corredata di un'interfaccia utente user-friendly.

Semantica del problema per realizzazione UML

a) L'enumerativo ***Categoria*** il quale definisce i valori ANTIPASTO, PRIMO, SECONDO, DESSERT*;*

b) La classe ***Portata*** la quale ha come caratteristiche categoria del piatto, un codice identificativo, il nome del piatto e prezzo. Realizzare tutti i metodi d'accesso e il metodo toString(). Una portata deve essere confrontabile con un'altra utilizzando come criterio il codice identificativo.

c) La classe ***Menu*** rappresenta un menù con le rispettive caratteristiche, in particolare, un menù è caratterizzato da un nome e da una mappa con un elenco di portate per ogni categoria: la mappa è recuperabile tramite il metodo ***getPortate*** senza parametri, mentre la lista di portate per categoria è recuperabile mediante il metodo ***getPortate(Categoria)***. Oltra ai rispettivi metodi d'accesso è necessario disporre anche del metodo toString(). Al costruttore della classe viene passato solo il nome del menù ed esso ha il compito di inizializzare la mappa. L'inizializzazione consiste nel creare una chiave con tutte le categorie e associarle ad una lista inizialmente vuota. Deve essere possibile aggiungere una portata ad una categoria.

d) **La classe *Ordine*** rappresenta il menu ordinato da uno specifico cliente, ossia l'insieme delle portate da lui scelte. Le caratteristiche di un ordine sono un nome del cliente, un menù e una mappa per gestire l'elenco delle portate. La mappa ha come chiave una categoria associata una sola portata perché un ordine può contenere al più una portata per categoria (ad es., non è possibile ordinare due primi). Oltre agli opportuni d'accesso, la classe espone i metodi:

- ***Ordine*** (costruttore), i cui argomenti sono il menù a cui ci si riferisce e il nome del cliente: entrambi gli argomenti non possono essere nulli e il nome del client non può essere la stringa vuota o una stringa composta da soli spazi;

- ***getElencoPortate***, che restituisce la mappa che mette in corrispondenza una categoria con la corrispondente portata ordinata

- ***aggiungiPortata***, che aggiunge una portata all'ordine: nel caso sia già presente una portata della stessa categoria, occorre lanciare una ***Exception***

- ***getNomeCliente***, restituisce il nome del cliente specificato nel costruttore

- ***sostituisciPortata***, che cambia una portata (primo argomento) con un'altra (secondo argomento) della stessa categoria*;* se l'operazione fallisce (se la portata da sostituire non è presente, se si tenta di sostituire una portata di una certa categoria con una portata di un'altra categoria o se la nuova portata non è presente nel menù di riferimento) si deve lanciare una ***Exception*** con messaggio d'errore adeguato alla circostanza;

- ***isValid***, che verifica se l'ordine contiene una portata per ogni categoria;

- ***getPrezzoTotale***, che restituisce il prezzo totale dell'ordine configurato in base alle portate scelte;

- ***toString***, che produce una rappresentazione compatta (solo nome e prezzo totale);

- ***toFullString***, che produce una rappresentazione estesa (nome, prezzo totale ed elenco delle portate.

Persistenza su file

Come anticipato, Portate.txt e Menu.txt contengono rispettivamente le portate disponibili e i menù già predisposti.

Nel file Portate.txt ogni riga contiene nell'ordine codice univoco, descrizione, categoria e prezzo, separati da virgole:

```
Formato del file Portate.txt
A01, Prosciutto e melone, ANTIPASTO, 9.00
A02, Antipasto di mare, ANTIPASTO, 8.00
...
```

Per la gestione delle portate è opportuno realizzare l'interfaccia **PortateManager** la quale espone i seguenti metodi:

public HashMap<Categoria, ArrayList<Portata>> caricaPortate();
public void salvaPortata(Portata p);

Il primo metodo restituisce una mappa che indicizza le categorie associate alle portate di quella categoria lette dall'apposito file mentre il secondo metodo aggiunge al file delle portate una nuova portata a quelle già esistenti.

Tale interfaccia deve essere implementata dalla classe **MyPortateManager.**

Il file Menu.txt contiene invece una serie di record che descrivono i vari menù: la prima riga contiene la parola chiave MENU seguita dal nome del menu, mentre le righe successive elencano le portate fra cui si può scegliere nelle diverse categorie (una categoria per riga), riportandone i codici univoci separati da virgole, come nell'esempio sotto; il record è poi chiuso dalla riga contenente "END MENU".

```
Formato del file Menu.txt
MENU base
ANTIPASTI: A01, A02
PRIMI: P02, P04
SECONDI: S03, S05
DESSERT: D01
END MENU
...
```

Per la gestione dei menù è opportuno realizzare l'interfaccia **MenuManager** la quale espone il seguente metodo:

public ArrayList<Menu> caricaMenu(Map<Categoria, List<Portata>>mm);
public void salvaMenu(Menu m);

Il primo metodo restituisce la lista di menù letti da file mentre il secondo provvede a salvare un nuovo menù su file.

Tale interfaccia dovrà essere implementata dalla classe **MyMenuManager.**

Nota 1: in ***MyPortateManager*** per trasformare il token che rappresenta una categoria in un'istanza dell'enumerativo ***Categoria*** è opportuno usare il metodo statico ***valueOf*** degli enumerativi; nel caso in cui la stringa non corrisponda al nome di un valore dell'enumerativo, il metodo lancia una ***Exception***. Nella mappa restituita deve essere presente una entry per ogni categoria - dove, al limite, la lista corrispondente è vuota.

Nota 2: in ***MyMenuManager*** per popolare l'elenco di portate per categoria di un ***Menu*** occorre recuperare la lista di portate per la categoria mediante il metodo ***Menu.getPortate(Categoria)***, poi popolare direttamente la lista ricevuta, che è la lista interna alla classe ***Menu***. Un esempio d'uso è riportato di seguito:

```
Menu m = ...;
Portata p = ...;
List<Portata> portatePerCategoria =
m.getPortate(Categoria.ANTIPASTI);
portatePerCategoria.add(p);
```

SUGGERIMENTO: per la realizzazione della scrittura, si suggerisce di aggiungere metodi disponibili nella classe ***Portate e Menu*** (in particolare *toFileFormat)*.

Il Controller

L'obiettivo dell'applicazione è permettere al cameriere (o al cliente dotato di tablet) di confezionare l'ordine di un cliente, partendo dal menu prescelto ed effettuando al suo interno le scelte fra le diverse portate offerte. Inoltre l'applicazione deve consentire di agigungere una nuova portata e un nuovo menu.

È necessario realizzare un'interfaccia **Controller** devono essere presenti i seguenti metodi con le firme riportate sotto:

String sostituisciPortata(Ordine ordine, Portata daMettere);
ArrayList<Menu> getMenus();
Ordine creaOrdine(Menu m, String nomeCliente);
boolean aggiungiPortata(Portata p);
boolean aggiungiMenu(Menu m);

La classe *MyController* deve essere realizzata *e* implementa l'interfaccia ***Controller.***
Il costruttore riceve come argomenti un PortataManager, un MenuManager e uno ***UserInteractor***, provvede al popolamento delle strutture dati necessarie; in caso di errore nella lettura del file, notifica l'utente e termina l'applicazione (catturare le opportune eccezioni).

UserInteractor è un'interfaccia che espone i metodi:

void showMessage(String message);
void shutDownApplication();

Per terminare l'applicazione, usare il metodo ***shutDownApplication*** di ***UserInteractor.***

Il metodo ***getMenus*** restituisce un ArrayList di ***Menu*** (ottenuta nel costruttore mediante i reader).

Il metodo **creaOrdine** crea un Ordine partendo dal Menu e dal nome del cliente passati come parametro, usando l'opportuno costruttore di Ordine.

Il metodo **sostituisciPortata** effettua la sostituzione, nell'Ordine passato come primo argomento, della Portata passata come secondo argomento: la portata da sostituire viene determinata cercando nell' Ordine passato la portata correntemente selezionata con la categoria della portata passata. Il metodo restituisce una stringa null in caso di successo, o contenente il messaggio opportuno in caso d'errore: in particolare, tale messaggio è ottenuto catturando le opportune eccezioni, e recuperando il relativo messaggio.

I metodi **aggiungiPortata** e **aggiungiMenù**, prendendo come parametro il rispettivo oggetto, hanno il compito di aggiungere al file il nuovo oggetto.

UserInteractor è un'interfaccia che espone i metodi:

void showMessage(String message);
void shutDownApplication();

Tale interfaccia viene implementata nella classe **SwingUserInteractor fornita.**

Interfaccia Utente

Appena l'applicazione parte, vengono caricati dal file Menu.txt tutti i menù e dal file Portate.txt tutte le portate: un eventuale errore di formato nei file è mostrato tramite un'opportuna finestra di dialogo, nel qual caso l'applicazione esce immediatamente senza neanche mostrare la GUI.

Se il caricamento preliminare ha esito positivo, compare la finestra principale dell'applicazione.

La GUI (**classe Risto*GUI)*** consente di:

- scegliere un menù fra quelli disponibili in una combobox, come risultato, vengono popolate le successive combo delle portate presenti nel menù, divise per categorie e viene calcolato il prezzo;

- Modificare una portata fra quelle di una data categoria, sostituendo quella preselezionata nella combo con un'altra: la modifica causa naturalmente l'aggiornamento del prezzo dell'ordine;

- cambiare idea e selezionare un altro menù, con conseguente ri-aggiornamento generale e ri-popolamento delle combo e ricalcolo del prezzo.

- Inserire una nuova portata specificando un form per raccoglierne i dati.

- Inserire un nuovo menù, scegliendo con delle combobox tra le portare esistenti.

Tutte le operazioni sui dati devono essere effettuate mediante il ***Controller*** ricevuto in ingresso dal costruttore.

SUGGERIMENTO : Utilizzare solo il controller tramite tra elaborazioni e interfaccia utente.

DA CONSEGNARE : 18 maggio 2017