Progetto Laboratorio Interdisciplinare A

Nguyen Manuel 741939 Varese
Digvijaysinh D. Raj 741976 Varese

Struttura Progetto

La struttura dell'applicazione è divisa in due parti: **clienti** e **ristoratori**.

Il package **clienti** è composto da:

Clienti.java

Entry principale dell'applicazione in cui si gestisce l'interfaccia con l'utente (**CLI**).
In questa classe viene chiesto all'utente che cosa vuole fare con la possibilità di scegliere tra:

1. Accesso Libero
2. Login
3. Registrazione

L'input è gestito dall'oggetto **Scanner** mentre le varie scelte tramite uno **switch-case** con possibilità di ritorno al menù iniziale premendo 1.

accessoLibero() è un metodo della classe Utente che permette di effettuare una ricerca in base al comune,tipologia e nome di un ristorante.

userLoggato() è un metodo della classe Login che gestisce l'autenticazione dell'utente chiedendo in input nickname e password, consente di accedere alle funzioni di accesso libero e anche di effettuare giudizi.
Ogni giudizio deve contenere obbligatoriamente un voto che va da 1 a 5 e opzionalmente un commento di lunghezza massima di 256 caratteri.

registraCliente() è un metodo della classe Registrazione che si occupa della registrazione di nuovi utenti nel sistema.

Registrazione.java

La classe Registrazione permette, una volta chiesti in input i dati ed effuttati gli opportuni controlli, di registrare l'utente su file qual'ora non fosse già registrato.

Iniziliazzata tramite un costruttore vuoto in quanto gli attributi verranno inizializzati tramite l'identificatore ***this***.

registraCliente()

Vengono chiesti all'utente i seguenti parametri (che appartengono alla classe):

- Nome
- Cognome
- Comune di residenza
- Sigla provincia
- Email
- Nickname
- Password

Sono presenti i vari metodi get degli attributi.

L'input Nome, Cognome e Comune di residenza verranno validati solo se hanno una lunghezza > 0.

Sigla provincia è valida solo se è composta da lunghezza uguale a 2.

Email è valida se

1. *isValidEmailAddress(String email)* ritorna true
2. check(String email, int parametro) ritorna false

Nickname è valido se

1. check(String nickname, int parametro) ritorna false
2. Lunghezza > 0

Password è valida se la lunghezza è >= 8.

check(String str, int parametro)

Il prototipo accetta:
- La stringa str da controllare
- L'intero parametro che rappresenta la posizione nel file

IL metodo legge il file '*Utenti.dati*' e per ogni riga la splitta tramite *stringa.split(",")* ritornando un array di String contenente in ogni i-esima posizione il valore rispettivo.
In seguito, in base a str e al parametro viene effettuato il controllo desiderato (nickname/email) e ritorna true se str è già presente nel file.

isValidEmailAddress(String email)

Il prototipo accetta:
- La stringa email da validare

Il metodo controlla che l'email sia in formato valido.

toString()

Il metodo concatena tutti i dati in una stringa separandoli da una ',' che faciliterà il processo di lettura nella altre classi

scriviFile()

Tale metodo viene richiamato una volta che sono stati chiesti e validati tutti i dati in input.
Riporterà sul file '*Utenti.dati*' tutti i dati relativi all'user.

Viene riportato in seguito un esempio

Giorgio,Rossi,Roma,RM,giorgio.rossi@gmail.com,giorgio,12345678

Utente.java

La classe Utente gestisce tutte le funzionalità di un utente non loggato nonché la ricerca per:

- Comune
- Tipologia
- Nome

La classe ha come unico attributo la String rist che conterrà il ristorante scelto dall'utente in base alla posizione desiderata.

E' presente un costruttore che inizializza String rist a *null*.

ricercaComune(String obj)

Il prototipo accetta una stringa obj che rappresante il Comune da ricercare nel file.
Il metodo legge la riga i-esima ed eseguendo opportunamente le split attua un confronto con l'obj e il comune letto.
Se il confronto risulta true, tale riga del file viene aggiunta all'**ArrayList<String>** che verrà usato come ritorno.

ricercaTipologia(String obj)

Il prototipo accetta una stringa obj che rappresante la tipologia da ricercare nel file.
Il metodo legge la riga i-esima ed eseguendo opportunamente le split attua un confronto con l'obj e la tipologia letta.
Se il confronto risulta true, tale riga del file viene aggiunta all'**ArrayList<String>** che verrà usato come ritorno.

ricercaNome(String obj)

Il prototipo accetta una stringa obj che rappresante il nome da ricercare nel file.
Il metodo legge la riga i-esima ed eseguendo opportunamente le split attua un confronto con l'obj e il nome letto.
Se il confronto risulta true, tale riga del file viene aggiunta all'**ArrayList<String>** che verrà usato come ritorno.

E’ stato deciso di utilizzare 3 funzioni diverse per le ricerche in quanto dopo una riflessione legata alla progettazione dell’applicativo si è compreso che fosse più opportuno in quanto venivano utilizzati diversi formati di inserimento.
Ogni riga contiene i vari dati divisi da una virgola però a sua volta l’indirizzo è diviso con i trattini.

selezionaRistorante(ArrayList<String> l, int x)

Il prototipo accetta un **ArrayList<String>** contenente i ristoranti e x rappresenta l’indice della posizione desiderato dall’utente.
Questo metodo richiama visualizzaInformazioni(String rist)

visualizzaInformazioni(String rist)

La Stringa rist contiene il ristorante scelto dall’utente.
Richiamo la funzione leggiGiudizi() che ritorna un **ArrayList<String>** contente tutti i giudizi relativi ad esso.
Stampa i dati relativi a rist.
Viene effettuato il conteggio per ogni voto (da 1 a 5).
Se sono presenti giudizi vengono stampati:
- Nickname
- Voto
- Commento (se presente)

leggiGiudizi()

Tale metodo scansiona il file ‘*EatAdvisor.dati*’.
Viene letta ogni riga del file e una volta che viene accertato che sia un commento tramite ‘G:’ viene effettuato un controllo sul nome del ristorante e in seguito viene salvato (e infine ritornato) nell’**ArrayList<String>** nickname,voto e commento.

accessoLibero()

E’ il metodo che gestisce il menù dell’utente non loggato chiedendogli quale ricerca vuole eseguire tra: Comune,Tipologia e Nome.
Per ogni ricerca vengono mostrati a video tutti i ristoranti trovati.
Infine viene chiesto all’utente quale ristorante vuole selezionare (tramite un indice intero che parte da 0 a lenght-1) riportando così i dettagli specifici di esso.
Da questo metodo partono tutte le funzioni utili riportate precedentemente.
Il tutto gestito viene gestito da un **switch-case**.

Login.java

La classe Login, permette di gestire tutte le funzionalità extra riservate agli utenti registrati ed estende la classe utente ereditandone i metodi e attributi principali.

Definito l’attributo String user che rappresenta l’utente loggato.

Il costruttore non accetta parametri però al suo interno richiama il costruttore padre tramite il metodo super().

login()

Questo è il metodo che si occupa dell’autenticazione dell’utente chiedendo in input nickname e password.
Viene scandito il file ‘*User.dati*’ e per ogni riga vengono confrontati nickname e password.
Se viene trovato il nickname e la password è corretta l’utente sarà loggato (salvandolo in user) e la funzione ritornerà true, false altrimenti.

giudica()
Questo metodo permette l’inserimento opzionale di un giudizio relativo ad un ristorante.

Viene chiesto all’utente se vuole esprimere un giudizio, se l’utente conferma gli viene chiesto il voto esprimibile da 1 a 5 (con relativo controllo).
Su decisione dell’utente può essere inserito un commento di lunghezza massima di 256 caratteri. Una volta che l’utente decide di inserire un commento esso non può essere vuoto.
Infine si riporta il giudizio richiamando la funzione scriviFile(int stelline, String commento, String rist)

scriviFile(int stelline, String commento, String rist)
In questa parte viene registrato il giudizio espresso precedentemente dall’utente sul file ‘*EatAdvisor.dati*’

Viene riportato in seguito un esempio

G:Arooooon,La Grottesca,via-Gramsci-35-Legnano-MI-20025,4,Ottimo cibo!

void userLoggato()
Quest’ultima funzione, che viene richiamata nel main dell’applicativo, svolge la funzione di autenticazione richiamando login() che se ritorna true richiamerà accessoLibero() e giudica()

Per quanto riguarda la parte dei ristoratori si è deciso di suddividere in classi Tipologia e Indirizzo mentre i restanti attributi appartengono a Ristorante.

Il package ristoratori è composto da:

Tipologia.java

Tipologia è la classe che gestisce la tipologia di ogni ristorante che può essere:

- Italiano
- Etnico
- Fusion

Ha un attributo String tipo che conterrà la tipologia scelta dall'utente.

Il costruttore accetta come parametro una String tipo che permetterà di inizializzare il tipo della classe.

Sono presenti i metodi get e set relativi al tipo

toString()

Metodo che ritorna la Stringa tipo

Indirizzo.java

Indirizzo è la classe che gestisce l'indirizzo di un ristorante, diviso in:
- qualificatore (via,viale,corso,piazza,piazzale)
- nome
- numeroCivico
- comune
- siglaProvincia
- Cap

E' presente il costuttore a cui vengono passati come parametri i dati elencati poco fa.

Sono presente le get e set degli attributi.

toString()
E’ il metodo che ritorna la Stringa contente tutti i dati relativi al ristorante dividendoli tramite ‘-’.

Viene riportato un esempio in seguito.

via-Roma-36-Varese-VA-21100

Ristorante.java
Ristorante è la classe che gestisce i dati relativi al ristorante tra cui:

- Nome del ristorante
- Indirizzo
- Numero di telefono
- URL sito
- Tipologia

In seguito provvede alla registrazione del ristorante su file.
I dati elencati sono di tipo String tranne per Indirizzo e Tipologia che sono rispettivamente di tipo Indirizzo e Tipologia.

Viene inizializzato un costruttore vuoto.

Sono definiti i vari metodi get e set degli attributi.

registraRistorante()
Questo metodo si occupa della richiesta in input dei vari dati del ristorante.

Il nome del ristorante non può essere vuoto o contenere una virgola o i due punti (perché andrebbe in conflitto con il formato dei dati presenti nel file).

I campi dell’indirizzo vengono controllati in questo modo:

Il qualificatore corrisponde a: via,viale,corso,piazza,piazzale.

Il nome del ristorante, numero civico non possono essere vuoti.

La sigla della provincia deve contenere esattamente 2 caratteri.

Il CAP deve essere formato da 5 numeri e deve essere validato tramite la funzione isNumeric(String strNum).

Il numero deve avere dalle 6 ai 11 cifre e deve anch’esso essere validato da isNumeric(String strNum).

L’URL del sito non può essere vuoto.

A questo punto viene creato ed inizializzato l’oggetto Indirizzo.

La tipologia deve per forza essere una tra:

- Italiano
- Etnico
- Fusion

Viene creato e inizializzato l’oggetto Tipologia.

isNumeric(String strNum)

Questo funziona controlla che la String strNum contenga solo numeri.

toString()

E’ il metodo che concatena tutti i dati del ristorante ritornandoli nella stringa *nomeRistorante* separandoli da ‘,’

checkRistorante()

Tale funzione scandisce il file ‘*EatAdvisor.dati*’ e per ogni riga, con le dovute split, controlla che non sia stato già inserito un ristorante avente lo stesso nome e lo stesso indirizzo.
Ritorna true se presente, false altrimenti.
Un ristorante con un nome già presente ma con un indirizzo diverso è conforme.
Una volta che sono stati inseriti tutti i dati e validati si procede con la scrittura su file di essi.

scriviFile()

Il metodo riporta la stringa ritornata da toString() che rappresenta l’oggetto Ristorante sul file ‘EatAdvisor.dati’.
All’inizio della riga viene inserita una ‘I:’ per separare l’inserimento di un ristorante dall’inserimento di un giudizio che è rappresentato dalla ‘G:’ iniziale.

Viene riportato in seguito un esempio

```
I:L'Etnico,via-Picasso-67-Olgiate olona-VA-21057,3893669667,www.etnico.it,Etnico
```

Ristoratori.java

Ristoratori è la classe contenente l’entry principale dell’applicativo che crea una nuova istanza della classe Ristorante tramite l’identificatore **new** e avvia il metodo registraRistorante() che permette di registrare un nuovo ristorante salvandolo su un file con estensione .dati