

Eat Advisor

MANUALE TECNICO

Installazione	P. 1
1. Installazione	P. 1
Archiviazione dati	P. 1
1. Tipo dei files	P. 1
2. Strutture dati	P. 1
Struttura	P. 2
1. Struttura progetto	P. 2
2. Struttura App Clienti	P. 3
3. Struttura App Ristoratori	P. 4
Scelte implementate e algoritmiche	P. 4
1. Riutilizzo del codice	P. 4
2. Complessità	P. 5

INSTALLAZIONE

1. INSTALLAZIONE

Una volta installate le applicazioni per clienti e ristoratori, al primo avvio creano automaticamente i file di dati “Utenti.dat” e “EatAdvisor.dat” al percorso <user.home>/EatAdvisor/. L'applicativo per ristoratori crea, se non esiste già, solo il file “EatAdvisor.dat”, mentre l'applicativo per clienti genera entrambi i files (o solo “Utenti.dat” nel caso “EatAdvisor.dat” sia già stato creato dall'applicativo per ristoratori). Entrambi i files vengono caricati coi dati di default richiesti in consegna, all'avvio.

ARCHIVIAZIONE DATI

1. TIPO DEI FILES

I dati degli utenti, dei ristoranti, i giudizi e tutte le altre informazioni utili al funzionamento dei programmi (quali indirizzo, tipologia, ecc...), vengono salvate sui files “Utenti.dat” e “EatAdvisor.dat”. Questi due documenti sono documenti di oggetti. Sono stati preferiti ai files di testo perché più versatili in fase di ricerca ed estrazione. Una volta trovato il dato cercato e opportunamente castato, l'oggetto è pronto per essere utilizzato. Questa operazione sarebbe stata più lenta coi files di testo e avrebbe richiesto operazioni di maggiore complessità computazionale. Per cui le classi Cliente, Ristorante, Giudizio e Indirizzo sono tutte serializzabili.

1. STRUTTURE DATI

Le strutture dati utilizzate per lo sviluppo dei programmi sono files di oggetti, vettori e liste dinamiche. L'implementazione delle liste

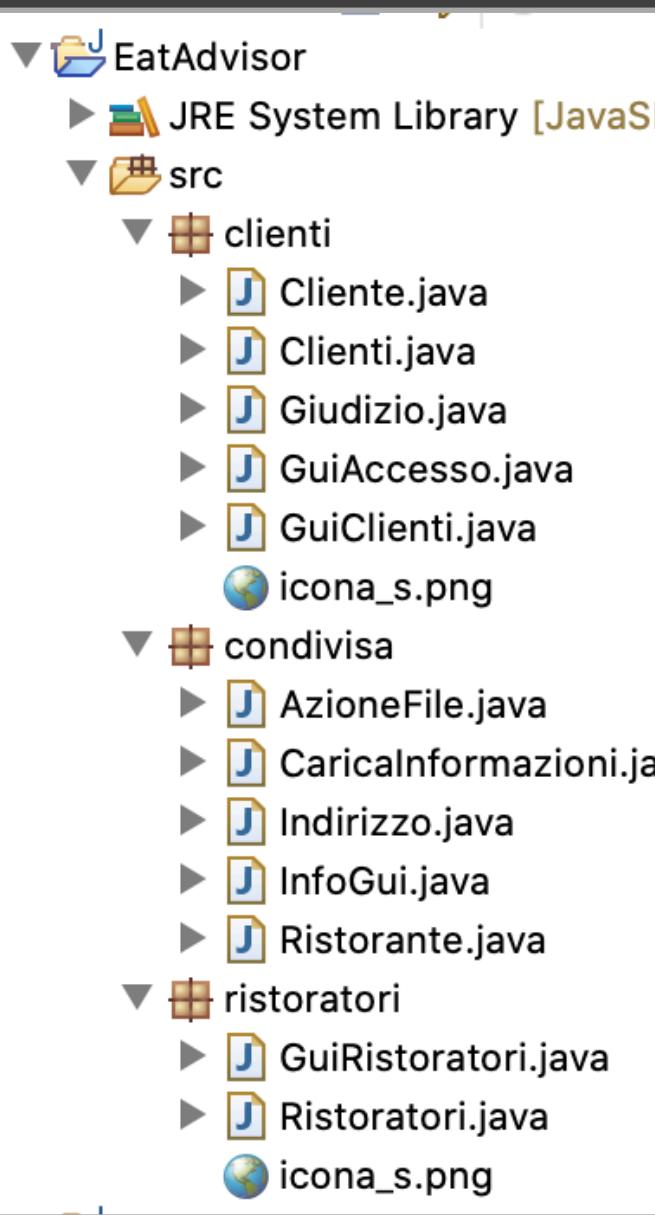
dinamiche è avvenuta attraverso ArrayList perché più efficienti in termini di velocità e spazio occupato rispetto alle liste di tipo Vector. In fase di ricerca, inserimento o cancellazione di un oggetto, i dati vengono temporaneamente salvati su un ArrayList sul quale vengono svolte le operazioni richieste.

STRUTTURA

1. STRUTTURA PROGETTO

Il progetto è suddiviso in tre package: clienti, ristoratori e condivisa. Nel primo vi sono tutte le classi necessarie al funzionamento del programma clienti, nel secondo quelle al programma ristoratori e nel terzo le classi e le risorse utilizzate da entrambi gli applicativi. Nel package “condivisa”, oltre alle classi “Indirizzo” e “Ristornate”, ci sono classi che mettono a disposizione funzioni base utili agli applicativi per clienti e ristoratori.

“InfoGui” si occupa di gestire le proporzioni della GUI degli applicativi rispetto alla grandezza dello schermo, “AzioneFile” offre metodi utili per l’interazione tra applicazione e i file di dati di oggetti (scrittura, sovrascrittura, ricerca, ricerca per tipo, cancellazione, ecc...). La classe “CaricalInformazioni” si occupa di inserire nei files i dati di default di clienti, ristoranti e giudizi richiesti dalla consegna. La struttura dell’applicazione per clienti e quella dell’applicazione per ristoratori sono descritte dettagliatamente nei prossimi punti.



2. STRUTTURA APP CLIENTI

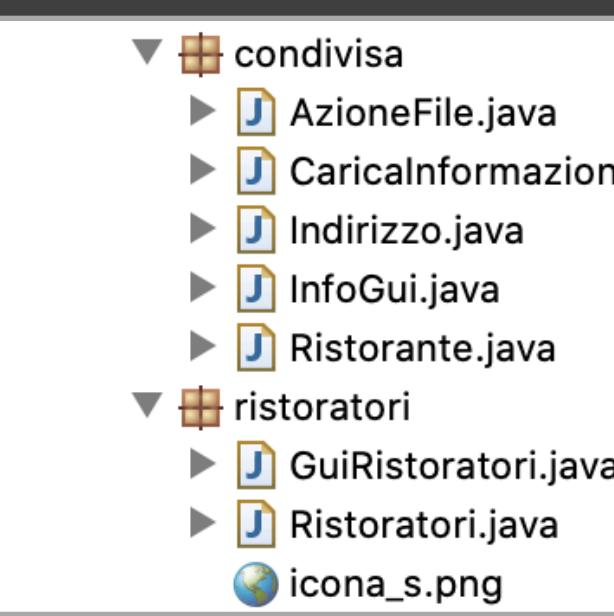
L’applicazione clienti è costituita dalle classi presenti nel package “clienti” e quelle del package “condivisa”.

- **Cliente:** è la classe che descrive i clienti che potranno accedere al programma.
- **Clienti:** è la classe principale contenente il main dell’applicazione clienti. All’apertura avvia “GuiAccesso” e carica i dati di default attraverso “CaricalInformazioni”.
- **Giudizio:** è la classe che descrive i giudizi espressi dai clienti riguardo ad un ristorante.
- **GuiAccesso:** è la classe contenente la GUI per l’accesso all’applicazione clienti. Dopo l’accesso viene avviata “GuiClienti”, che deve ricevere come parametro per il costruttore un cliente utilizzato per l’operazione di giudizio. Se l’accesso avviene come ospite viene passato “null”, se invece si accede con un’utenza valida o ci si registra, viene passato il cliente autenticato.
- **GuiClienti:** è la classe contenente la GUI dell’applicazione clienti dove è possibile svolgere le operazioni di ricerca sui ristoranti, di selezione e aggiungere i giudizi.
- **Indirizzo:** è la classe che descrive un indirizzo, utile per la classe “Ristorante”.
- **Ristornate:** è la classe che descrive i ristoranti sui quali è possibile svolgere le operazioni di ricerca e giudizio da “GuiClienti”.

▼	clienti
►	J Cliente.java
►	J Clienti.java
►	J Giudizio.java
►	J GuiAccesso.java
►	J GuiClienti.java
►	icona_s.png
▼	condivisa
►	J AzioneFile.java
►	J CaricalInformazioni.java
►	J Indirizzo.java
►	J InfoGui.java
►	J Ristorante.java

3. STRUTTURA APP RISTORATORI

- Indirizzo: è la classe che descrive un indirizzo, utile per la classe “Ristorante”.
- Ristorante: è la classe che descrive i ristoranti sui quali è possibile svolgere le operazioni di aggiunta, cancellazione o visione da “GuiRistoratori”.
- Giudizio: è la classe che descrive i giudizi espressi dai clienti riguardo ad un ristorante.
- GuiRistoratori: è la classe contenente la GUI per all'applicazione ristoratori dove è possibile svolgere le operazioni di aggiunta, cancellazione o visione dei ristoranti.
- Ristoratori: è la classe contenente il main dell'applicazione ristoratori e carica i dati di default attraverso “CaricalInformazioni”.



SCELTE IMPLEMENTATIVE E ALGORITMICHE

1. RIUTILIZZO DEL CODICE

Per l'interazione tra programma e files vi è una classe “AzioneFile” che mette a disposizione metodi base (lettura, scrittura, ricerca, cancellazione, ...) per entrambi i programmi di EatAdvisor, favorendo così il riutilizzo del codice.

```
public boolean scrivi(Object obj, boolean repeat) {
    if(!f.exists()) {
        System.out.println("Il file non esiste");
        return false;
    }
    System.out.println("Il file esiste > Scrittura");

    ArrayList<Object> keep = new ArrayList<Object>();
    if(f.length()>0) {
        keep = leggi(false);
        if(!repeat)
            if(trova(keep, obj))
                return false;
    }
    keep.add(obj);

    sovrascrivi(keep);
    return true;
}
```

AzioneFile

```
if(ristorante != null) {
    if(file.scrivi(ristorante, false)) {
        lbl_err.setText("Ristorante aggiunto.");
        txt_nome.setText(null);
        txt_ind.setText(null);
        txt_num.setText(null);
        txt_com.setText(null);
        txt_sig.setText(null);
    }
    file.scrivi(giudizio, false);
    txt_giu.setText("");
    cmb_val.setSelectedIndex(0);
    attivaDisattivaPannello(pnl_giud, false);
    caricaGiudizi(ristorante);
}
attivaDisattivaPannello(pnl_giud, false);
```

GuiRistoratori

GuiClienti

In alcuni casi, in GuiClienti e GuiRistoratori, alcuni dei seguenti metodi sono stati riscritti da zero e fatti ad hoc per garantire una maggiore efficienza ed evitare che il riutilizzo del codice finisse per risultare ridondante nei relativi adattamenti gravando sulla complessità computazionale.

2. COMPLESSITA'

Nello sviluppo del progetto, dove possibile, si è cercato di sviluppare algoritmi a complessità lineare o al massimo quadratica per garantire buone performance anche al crescere dei dati. Le interazioni coi files sono state semplificate facendo uso di ArrayList e operando direttamente su questi piuttosto che ciclare più volte all'interno dei files. Il contenuto dei files viene filtrato (in base allo scopo richiesto) e copiato all'interno delle liste, dove avvengono le operazioni per fare alla fine il procedimento inverso: la scrittura dalle liste ai files.