**Manuale Tecnico progetto TheKnife**

* **Autori**

Artur Mukhin

De Giorgi Filippo

Magrin Nicolò

Caredda Anna Eleonora

* **Data =** 26/06/2025
* **Versione Documento=** v1.3

**Indice**

1. **Architettura dell'Applicazione**

**1.1.** Panoramica Generale e Struttura dei Package

**1.3.** Scelte Architetturali

1. **Strutture Dati e Gestione dei File**

**2.1.** Entità

**2.2.** Gestione della Persistenza

**2.3.** Formato dei File Dati

1. **Dettaglio delle Classi**

**3.1.** classe TheKnife

**3.2.** menu.GestioneMenu

**3.3.** repository.RistoranteService

**3.4.** gestioneFile.FileRistorante

**3.5.** entita.Ristorante

**3.6.** eccezioni.LocaleGiaPresenteException

1. **Funzionalità e Scelte Algoritmiche**

**4.1.** Flusso di Esecuzione Principale

**4.2.** Ricerca Ristoranti

**4.3.** Gestione Recensioni

**1. Architettura dell'Applicazione**

**1.1. Panoramica Generale e Struttura dei Package**

L'architettura del software è stata progettata per essere modulare e stratificata, separando le responsabilità in diversi package.

* **theknife**: Contiene la classe TheKnife, punto di ingresso dell'applicazione.
* **menu**: Gestisce l'interfaccia utente da riga di comando. Le classi \*UI e GestioneMenu si occuoano dell’interazione con l'utente e invocano i servizi appropriati.
* **repository**: Le classi Service fungono da intermediari tra l'interfaccia utente e il layer di accesso ai dati.
* **gestioneFile**: gestisce la lettura e la scrittura degli oggetti da e verso i file di testo.
* **entità**: Modella gli oggetti principali dell’applicazione (es. Utente, Ristorante, Recensione).
* **eccezioni**: Contiene le eccezioni personalizzate per la gestione di errori specifici dell'applicazione.

**1.2. Scelte Architetturali**

* **Uso di classi Service**: rende il codice della UI più snello e focalizzato sulla presentazione. Questa scelta promuove la riusabilità del codice e facilita i test.
* **GestioneFile**: astrae il modo in cui i dati vengono salvati. Questo significa che in futuro si potrebbe sostituire la gestione tramite file con un database senza dover modificare i layer di servizio e di presentazione.
* **Ereditarietà**: Le classi Cliente e Gestore ereditano dalla classe Utente, condividendo attributi comuni come username e password ma differenziandosi per il ruolo. Questa scelta riduce la duplicazione del codice.

**2. Strutture Dati e Gestione dei File**

**2.1. Entità Principali**

Le principali strutture dati utilizzate sono le classi del package entita e le collezioni per gestirle in memoria.

* **Utente (e sottoclassi Cliente, Gestore)**: Memorizza i dati anagrafici e di accesso degli utenti. Il Ruolo distingue tra cliente e gestore.
* **Ristorante**: Contiene tutte le informazioni relative a un ristorante, come nome, indirizzo, tipo di cucina e servizi offerti.
* **Recensione**: Modella una recensione, associandola a un utente, un ristorante, un punteggio in stelle e un testo.
* **PreferitiCliente**: Associa un usernameCliente a una List<Ristorante>, rappresentando la lista dei ristoranti preferiti di un cliente.
* **AssGestoreRistoranti**: Associa un usernameRistoratore a una List<Ristorante>, rappresentando l'elenco dei ristoranti gestiti da un ristoratore.

**2.2. Formato dei File**

La persistenza dei dati è gestita tramite file di testo. Le classi nel package gestioneFile sono responsabili dell’aggiunta, rimozione e modifica degli oggetti da e verso questi file. La scelta di un formato testuale permette una facile ispezione e modifica manuale dei dati a scopo di debug e test.

**3. Dettaglio delle Classi**

**3.1. Package entita**

* **Utente.java**: classe base che modella l’utente, cliente.java e Gestore.java la estendono ereditandone le proprietà, come modificatore di accesso è stato usato “protected” in quanto permette solo a sottoclassi e classi nello stesso package di accedere in modo diretto ai campi.
* **Cliente.java**: Rappresenta un utente con il ruolo di "cliente". Eredita da Utente ma scandisce il ruolo dell’utente che fa l’accesso.
* **Gestore.java**: Rappresenta un utente con il ruolo di "ristoratore". Eredita da Utente e serve a distinguere questa tipologia di utente per l'associazione con i ristoranti gestiti.
* **Ristorante.java**: Classe centrale che incapsula tutti gli attributi di un ristorante (nome, luogo, prezzo, cucina, servizi, ecc.). Include metodi getter e setter per ogni attributo.
* **Recensione.java**: Contiene i dati di una recensione: l’username dell'autore, le stelle (da 1 a 5), un testo descrittivo, la data e il nome del ristorante recensito.
* **PreferitiCliente.java**: Classe che lega l’username di un cliente a una lista di oggetti Ristorante, implementando la funzionalità di "preferiti".
* **AssGestoreRistoranti.java**: Classe che associa l’username di un ristoraratore a una lista dei Ristorante che gestisce.
* **Ruolo.java**: Classe enumerativa che definisce i due possibili ruoli per un utente: gestore e cliente.

**3.2. Package eccezioni**

* **LocaleGiaPresenteException.java**: È una RuntimeException personalizzata. Viene sollevata quando un Gestore tenta di inserire un nuovo ristorante che è già presente nel sistema.

**4. Funzionalità e Scelte Algoritmiche**

**4.1. Flusso di Esecuzione Principale**

1. L'esecuzione parte dal main in TheKnife.java.
2. Viene creato un oggetto GestioneMenu che mostra il menu iniziale.
3. L'utente sceglie un'opzione (es. "cerca ristorante").
4. GestioneMenu chiama il metodo appropriato della classe UI corrispondente (es. RistoranteUI.cercaRistorante()).
5. La classe UI interagisce con l'utente per raccogliere i parametri necessari (es. il luogo per la ricerca).
6. La classe UI invoca il metodo del servizio corrispondente
7. Il servizio chiama la classe di gestione file per ottenere tutti i dati.
8. Il servizio esegue la filtrazione dei ristoranti.
9. Il risultato viene restituito alla UI, che si occupa di mostrarlo all'utente.

**4.2. Ricerca Ristoranti**

L'algoritmo carica l'intera lista di ristoranti in memoria. Successivamente, applica uno dopo l'altro i filtri richiesti dall'utente, riducendo progressivamente la lista dei risultati, per poi mostrarli all’utente in lista.

**4.3. Gestione Recensioni**

L'aggiunta, modifica o cancellazione di una recensione viene gestita nel seguente modo:

1. L'intera lista di recensioni viene letta dal file recensioni.csv.
2. L'operazione desiderata viene eseguita sulla lista in memoria.
3. L'intera lista aggiornata viene riscritta sul file, sovrascivendo i contenuti precedenti.

**4.4. Aggiunta Ristorante (Gestore)**

Un gestore può aggiungere un ristorante nuovo o "rivendicare" uno esistente non ancora associato.

RistoranteService inizialmente esegue una serie di controlli:

* 1. Verifica che il ristorante esista.
  2. Verifica che lo stesso gestore non lo possieda già.
  3. Verifica che il ristorante non sia già di proprietà di un altro gestore.
  4. Se i controlli passano, crea o aggiorna l'associazione nel file username\_ristoranti.csv.

**4.5. Registrazione e Login Utente**

* **Registrazione**: Gestita da UtenteUI.registrazione(). La UI raccoglie i dati dall'utente. Un controllo assicura che lo username inserito non sia già presente. Una volta validati i dati, viene creata un'istanza di Cliente o Gestore in base al ruolo scelto. L'oggetto viene quindi salvato nel file utenti.csv.
* **Login**: Gestita da UtenteUI.login(). La UI raccoglie username e password. La validazione avviene confrontando la password fornita con quella recuperata tramite il metodo utenteServ.get(username).getPassword(). Se le credenziali sono corrette, il metodo restituisce l’Utente completo, permettendo a GestioneMenu di avviare la sessione appropriata (cliente o gestore).