

MANUALE TECNICO

Università degli Studi dell'Insubria – Laurea Triennale in Informatica

Corso di Laurea: Laurea Triennale in Informatica

Laboratorio Interdisciplinare A (a.a. 2024/2025)

Docente

Prof. Loris Bozzato

Nome del progetto

TheKnife

Autori

Nome e Cognome	Matricola
Hadir Oueslati	762887
Mohamed Mounir Mechri	767786
Raja Sofia Neri	762483

SOMMARIO

- **INTRODUZIONE**
- **STRUTTURA GENERALE DEL CODICE**
 - Utente
 - Cliente
 - Ristoratore
 - GestioneDati
 - Menu
 - Recensione
 - Ricerca
 - Ristorante
 - TheKnife

INTRODUZIONE

1. Cos'è TheKnife?

"TheKnife" è un'applicazione software progettata per simulare una piattaforma digitale per la ricerca e gestione di ristoranti, simile a "TheFork".

Consente a utenti e ristoratori di interagire attraverso funzioni di consultazione, recensione e gestione.

Il progetto è stato realizzato per integrare competenze di programmazione, strutture dati e architettura degli elaboratori in un'applicazione reale

2. Librerie utilizzate

Per sviluppare il progetto TheKnife, abbiamo utilizzato le seguenti librerie:

- Java (versione recente - min 17 consigliato)
- JDK
- Apache Commons CSV (gestione file CSV)

STRUTTURA GENERALE DEL CODICE

TheKnife è un'applicazione software formata da nove classi principali:

- Utente
- Cliente
- Ristoratore
- GestioneDati
- Menu
- Recensione
- Ristorante
- Ricerca
- TheKnife

Utente.java

1 - Introduzione

La classe astratta *Utente* è una componente fondamentale del pacchetto "theknife".

Essa la classe base per tutti i tipi di utente che interagiranno con il sistema.

La sua finalità è quella di definire una struttura comune e incapsulare gli attributi e i metodi di base che ogni utente deve possedere, garantendo uniformità e coerenza nel design del software.

Essendo una classe abstract, non può essere istanziata direttamente, ma deve essere estesa da classi figlie come (es. *cliente*, *ristoratore*).

2. Struttura e Campi

La classe *Utente* contiene i seguenti campi, tutti definiti come `protected` per consentire l'accesso da parte delle classi che la estendono:

- **nome (String)**: Il nome di battesimo dell'utente.
- **cognome (String)**: Il cognome dell'utente.
- **username (String)**: Il nome utente univoco utilizzato per l'accesso.
- **passwordHash (String)**: La password dell'utente, memorizzata in forma di hash per motivi di sicurezza.
- **domicilio (String)**: L'indirizzo di residenza dell'utente.
- **ruolo (String)**: Il ruolo dell'utente all'interno del sistema (es. "client", "admin", ecc.).

3. Metodi e Funzionalità

La classe Utente offre due metodi pubblici per l'accesso ai dati dell'utente, noti come getter:

- **getUsername():** Restituisce il nome utente dell'oggetto Utente. È un metodo chiave per l'identificazione e l'autenticazione.
- **getRuolo():** Restituisce il ruolo dell'utente. Questo metodo è utile per controllare i permessi e le funzionalità a cui l'utente ha accesso nel sistema.

4. Costruttore

Il costruttore di Utente inizializza tutti i campi protetti.

La sua firma è la seguente:

```
public Utente(String nome, String cognome, String username,  
String passwordHash, String domicilio, String ruolo)
```

Questo costruttore deve essere chiamato dalle classi figlie (`super(...)`) per inizializzare gli attributi ereditati.

5. Relazioni e Progettazione

- **Ereditarietà:** La classe Utente è progettata per essere il superclasse di altre classi (ad esempio, Cliente e Ristoratore). Le classi figlie ereditano i campi protetti e i metodi pubblici, aggiungendo le proprie funzionalità specifiche.
- **Principio di Inerzia:** Nonostante Utente gestisca attributi essenziali, non contiene logica di business complessa o metodi per la modifica dei dati, mantenendo il suo ruolo di "contenitore" di dati di base e garantendo che le responsabilità specifiche siano delegate alle classi figlie.

Cliente.java

1. Introduzione

La classe Cliente estende la classe Utente e rappresenta un utente dell'applicazione con specifiche funzionalità legate alla gestione di ristoranti preferiti e recensione. L'obiettivo di questa classe è quello di incapsulare il comportamento di un cliente fornendo metodi per interagire con i dati dei ristoranti e delle recensioni in modo sicuro.

2. Architettura e Struttura delle Classe

La classe Cliente è definita all'interno del package "theknife". Eredita gli attributi e i metodi dal superclasse Utente, aggiungendo due campi privati per gestire le interazioni specifiche di un cliente:

***preferiti:** Una ArrayList<Ristorante> che memorizza i ristoranti che l'utente ha aggiunto alla lista dei preferiti.

***mierecensione:** Una Map<Ristorante, Recensione> che associa ogni ristorante recensito alla recensione specifica lasciata dal cliente.

Il costruttore Cliente (String nome, String cognome, String username, String passwordHash, String domicilio) serve per inizializzare un nuovo oggetto Cliente, passando i parametri al costruttore della superclasse Utente e impostando il ruolo su "client".

3. Metodi e Funzionalità


La classe Cliente offre un insieme di metodi per gestire le due funzionalità principali:

3.1 Gestione dei Ristoranti Preferiti (preferiti)

- **getPreferiti()**: Restituisce l'elenco dei ristoranti preferiti.
- **aggiungerePreferiti(Ristorante r)**: Aggiunge un ristorante all'elenco dei preferiti se non è già presente.
- **eliminarePreferiti(Ristorante r)**: Rimuove un ristorante dall'elenco dei preferiti.
- **visualizzarePreferiti()**: Stampa a console l'elenco dei ristoranti preferiti. Se la lista è vuota, notifica all'utente che non ci sono preferiti.

3.2 Gestione delle Recensioni (mierecensione)

- **getMieRecensione()**: Restituisce la mappa delle recensioni lasciate dal cliente.
- **lasciareRecensionir(Ristorante r, int stelle, String testo)**: Questo metodo è il cuore della gestione delle recensioni. Prima di aggiungere una nuova recensione, verifica se il cliente ha già recensito lo stesso ristorante (ignorando la distinzione tra maiuscole e minuscole nel nome). Se una recensione esiste già, ne impedisce l'aggiunta e informa l'utente. In caso contrario, crea un nuovo oggetto Recensione, lo aggiunge alla mappa mierecensione e lo passa anche al ristorante per aggiornare la sua lista di recensioni. Restituisce true in caso di successo e false in caso di fallimento.
- **visualizzareMieRecensione()**: Stampa a console tutte le recensioni lasciate dal cliente, mostrando il nome del ristorante e il contenuto della recensione.
- **eliminareRecensione(String nomResto)**: Cerca una recensione basandosi sul nome del ristorante. Se la trova, la rimuove sia dalla



mappa mi recensione che dalla lista di recensioni del ristorante stesso. Notifica l'utente se l'operazione ha avuto successo o meno.

- **modificareRecensione(String nomResto, int stelle, String testo):**
Permette di modificare una recensione esistente. Cerca la recensione per nome del ristorante, aggiorna il numero di stelle e il testo della recensione e restituisce true. Restituisce false se la recensione non viene trovata.

4. Relazioni e Dipendenze

La classe Cliente ha una forte dipendenza dalle classi Ristorante e Recensione, in quanto i suoi metodi interagiscono direttamente con oggetti di questi tipi.

Il suo funzionamento è strettamente legato all'implementazione di questi ultimi.

La relazione con la classe Utente è di ereditarietà, il che significa che Cliente eredita tutte le proprietà e i metodi pubblici e protetti di Utente.

5. Considerazioni per la Manutenzione

- **Robustezza del codice:** I metodi come lasciareRecensione e eliminareRecensione includono controlli di esistenza (es. if (!preferiti.contains(r))) e notifiche all'utente (System.out.println) che rendono il codice più robusto.
- **Gestione degli errori:** L'utilizzo di return true/false per i metodi di modifica e l'eliminazione fornisce un modo chiaro per il chiamante di gestire il successo o il fallimento delle operazioni.

Ristoratore.java

1. Introduzione

La classe Ristoratore estende la classe astratta Utente e rappresenta l'entità di un proprietario di ristorante all'interno del sistema. Il suo scopo principale è quello di gestire i ristoranti di proprietà, consentendo l'aggiunta di nuovi locali, la visualizzazione di quelli esistenti e la gestione delle recensioni ricevute, in particolare permettendo di rispondere a tali recensioni.

2. Struttura e Campi

La classe Ristoratore è definita nel package theknife. Oltre agli attributi ereditati da Utente, ha un campo specifico:

- **mieiristoranti(ArrayList<Ristorante>):** Una lista che contiene tutti gli oggetti Ristorante che appartengono a questo Ristoratore.

3. Metodi e Funzionalità

La classe Ristoratore offre i seguenti metodi pubblici per la gestione dei ristoranti e delle recensioni:

- **aggiungereRistorante(Ristorante r):** Aggiunge un nuovo ristorante alla lista mesRestaurants.
- **getMesRestaurants():** Restituisce la lista dei ristoranti gestiti dal ristorante. Questo metodo è utile per accedere ai dati dei ristoranti.
- **visualizzareMieiRistoranti():** Stampa a console un elenco dettagliato dei ristoranti di proprietà del ristorante. Per ogni ristorante, mostra il nome, e se ci sono recensioni, le elenca una per una, inclusa l'eventuale risposta del ristorante. Se non sono presenti ristoranti, stampa un messaggio di avviso.

- **rispondeRecensione(Ristorante r, int indexAvis, String risposta):**

Questo metodo permette di rispondere a una recensione specifica. Accetta come parametri un oggetto Ristorante, l'indice della recensione nella lista del ristorante e il testo della risposta. Se l'indice è valido, imposta la risposta per la recensione corrispondente. Se l'indice non è valido, stampa un messaggio di errore.

4. Relazioni e Progettazione

- **Ereditarietà:** Ristoratore estende Utente, ereditandone le proprietà di base (username, passwordHash, ecc.). Il costruttore imposta il ruolo su "ristoratore".
- **Composizione:** La classe Ristoratore "possiede" una lista di oggetti Ristorante, dimostrando una relazione di composizione. Questo significa che un'istanza di Ristoratore gestisce direttamente le istanze di Ristorante ad essa associate.

GestioneDati.java

1. Introduzione

La classe statica GestioneDati è la componente del progetto responsabile della persistenza dei dati. Il suo scopo principale è gestire il caricamento e il salvataggio di informazioni relative a ristoranti, utenti, recensioni e preferiti da e verso file CSV. L'uso di metodi static in questa classe indica che non è necessario creare un'istanza dell'oggetto per utilizzarne le funzionalità, rendendola una vera e propria utility.

2. Metodi e Funzionalità

La classe è organizzata in sezioni logiche che raggruppano i metodi in base alla tipologia di dati gestiti.

2.1 Gestione dei Ristoranti

- **public static List<Ristorante> caricaRistoranti(String chemin):**

Questo metodo legge i dati dei ristoranti da un file CSV specificato dal percorso chemin. Ogni riga del file, a partire dalla seconda (la prima è l'intestazione e viene ignorata), viene analizzata per creare un oggetto Ristorante. Gli attributi del ristorante vengono estratti dalla riga e convertiti nei tipi di dati appropriati (es. Double.parseDouble). In caso di errore durante la lettura del file, stampa un messaggio di errore a console e restituisce una lista vuota.

2.2 Gestione degli Utenti

- **public static List<Utente> caricaUtenti(String chemin):** Carica gli utenti da un file CSV. Legge ogni riga e, in base al valore del campo role, istanzia un oggetto Cliente o Ristoratore, aggiungendolo a una lista. Questo metodo gestisce la creazione dinamica di oggetti di tipo diverso a seconda del ruolo, mostrando un'applicazione pratica del polimorfismo.

2.3 Gestione dei Preferiti

- **public static void salvaPreferito(String username, Ristorante resto):** Salva un ristorante preferito aggiungendo una nuova riga al file data/favoris.csv. La riga contiene il nome utente del cliente e il nome del ristorante, separati da un punto e virgola. Utilizza la modalità di append (true) per non sovrascrivere i dati esistenti.
- **public static void caricaPreferiti(Cliente cliente, List<Ristorante> ristorante):** Legge il file data/favoris.csv e, per ogni riga che corrisponde al nome utente del client passato come parametro, cerca il ristorante corrispondente nella lista restaurants e lo aggiunge alla lista dei preferiti del cliente.

2.4 Gestione delle Recensioni

- **public static void salvaRecensione(String username, Ristorante resto, int stelle, String testo):** Salva una recensione in un file CSV chiamato data/avis.csv. Aggiunge una riga con il nome utente, il nome del ristorante, il numero di stelle e il testo della recensione. Sostituisce eventuali punti e virgola nel testo della recensione con virgole per evitare problemi nella formattazione del CSV.
- **public static void caricaRecensione(Cliente cliente, List<Ristorante> ristorante):** Legge il file data/avis.csv. Per ogni riga corrispondente al nome utente, crea un oggetto Recensione e lo aggiunge sia alla lista di recensioni del ristorante che alla mappa delle recensioni del cliente.

2.5 Aggiornamento dei File CSV

- **public static void aggiornamentoRecensioneCSV(List<Cliente> clients):** Riscrive l'intero file data/avis.csv con le recensioni aggiornate. Questo metodo è utile per gestire modifiche (come l'eliminazione o la modifica di una recensione) in modo persistente, assicurando che lo stato salvato rifletta l'attuale stato del sistema. L'uso di `false` nel `FileWriter` sovrascrive il contenuto del file.
- **public static void aggiornamentoPreferitiCSV(List<Cliente> clients):** Simile al metodo precedente, riscrive completamente il file data/favoris.csv con l'elenco dei preferiti aggiornato per ogni cliente. Questo approccio garantisce l'integrità dei dati.

3. Considerazioni per la Manutenzione

- **Gestione degli errori:** Tutti i metodi di caricamento e salvataggio sono racchiusi in blocchi try-catch, che gestiscono potenziali eccezioni (es. FileNotFoundException, IOException). Questo rende il programma più robusto, sebbene l'output dell'errore sia limitato a una stampa a console.
- **Sicurezza:** Le password sono gestite come "password hash", il che suggerisce un approccio corretto alla sicurezza, evitando di memorizzare le password in chiaro.
- **Dipendenze:** La classe ha una forte dipendenza dalle classi Ristorante, Utente, Cliente e Ristoratore, in quanto i metodi carica* e salva* interagiscono direttamente con gli oggetti di queste classi.
- **Scalabilità:** I file CSV sono un formato semplice per la persistenza dei dati, ma per un'applicazione su larga scala, sarebbe consigliabile utilizzare un database per una migliore gestione delle performance e delle transazioni.

Menu.java

1. Introduzione

La classe Menu è il punto d'ingresso principale dell'applicazione. Il suo ruolo è quello di gestire l'interfaccia a riga di comando, interagire con l'utente e coordinare le chiamate alle altre classi del progetto. Si occupa di caricare i dati all'avvio del programma, presentare i menu principali e secondari e gestire il flusso delle operazioni.

2. Struttura e Campi

La classe Menu contiene i seguenti campi privati per la gestione dello stato dell'applicazione:

- **scanner (Scanner)**: Un oggetto per la lettura dell'input dell'utente dalla console.
- **restaurants (List<Ristorante>)**: Una lista che memorizza tutti gli oggetti Ristorante caricati dal file ristoranti.csv.
- **utilisateurs (List<Utente>)**: Una lista che contiene tutti gli oggetti Utente caricati dal file utenti.csv.

Il costruttore di Menu si occupa di inizializzare le liste restaurants e utilisateurs richiamando i metodi di caricamento dalla classe GestioneDati.

3. Metodi e funzionalità

La classe Menu è suddivisa in vari metodi che gestiscono le diverse funzionalità dell'interfaccia utente.

3.1 Menu Principale

- **visualizzareMenuPrincipale():** Il metodo principale che avvia il ciclo di vita dell'applicazione. Presenta un menu con opzioni come "Ricerca dei ristorante", "login", "Registrazione" e "logout". Un ciclo while continua a mostrare il menu finché l'utente non sceglie di uscire.
- **visualizzareRecensione():** Legge e mostra tutte le recensioni dal file avis.csv, raggruppandole per ristorante. Se il file è vuoto o non esiste, stampa un messaggio appropriato. Le recensioni sono visualizzate in forma anonima.

3.2 Funzionalità per Utenti non Autenticati

- **ricercaRistorante():** Permette agli utenti non loggati di cercare ristoranti per città, utilizzando il metodo ricercaPerCitta della classe Ricerca. Stampa i risultati della ricerca.
- **login():** Gestisce il processo di autenticazione. Richiede username e password, calcola l'hash della password fornita dall'utente e lo confronta con quello salvato. In caso di successo, reindirizza l'utente al menu appropriato (menuClient o menuRistoratore) a seconda del suo ruolo.
- **registrazione():** Permette a un nuovo utente di registrarsi. Richiede i dati personali e il ruolo (cliente o ristorante), crea una riga con i dati e la scrive nel file utenti.csv.

3.3 Sottomenu del Cliente (*menuClient*)

Questo metodo gestisce l'interazione per un utente di tipo Cliente:

- **caricaPreferiti e caricaRecensione:** Prima di avviare il menu, carica i dati persistenti del cliente (preferiti e recensioni) dai file CSV tramite la classe GestioneDati.
- **Opzioni del menu:** L'utente può visualizzare i preferiti, aggiungerne o rimuoverne, gestire le proprie recensioni (aggiungere, visualizzare, eliminare, modificare), cercare ristorante e alla fine fare il logout e uscire dell'area riservata ai cliente .
- **Persistenza:** I metodi per aggiungere e rimuovere preferiti (GestioneDati.salvaPreferito e GestioneDati.updatePreferitiCSV) e per le recensioni (GestioneDati.salvaRecensione e GestioneDati.updateNoticeCSV) interagiscono direttamente con la classe GestioneDati per mantenere lo stato persistente.

3.4 Sottomenu del Ristoratore (*menuRistoratore*)

Questo metodo gestisce l'interazione per un utente di tipo Ristoratore:

- **caricaRistoranteDelProprietario:** Prima di avviare il menu, carica i ristoranti specifici del ristoratore leggendo il file `restaurants_proprietaires.csv`.
- **Opzioni del menu:** Il ristoratore può aggiungere un nuovo ristorante, visualizzare i suoi ristoranti con le relative recensioni e rispondere alle recensioni.
- **Persistenza:** L'aggiunta di un nuovo ristorante non solo lo aggiunge alla lista in memoria, ma lo salva anche nei file `ristoranti.csv` e `restaurants_proprietaires.csv`.

3.5 Logout

Serve per finire il programma e stampa il messaggio ("Arrivederci")

Recensione.java

1. Introduzione

La classe Recensione è un modello che rappresenta una recensione di un ristorante. Cattura i dettagli di una valutazione, come l'utente che l'ha scritta, il ristorante a cui si riferisce, il punteggio in stelle e il testo del commento. È un elemento fondamentale del sistema, che funge da oggetto di dati per le interazioni tra clienti e ristoranti.

2. Struttura e Campi

La classe contiene i seguenti campi privati che ne definiscono le proprietà:

- **utente (Utente)**: L'utente che ha scritto la recensione.
- **ristorante (Ristorante)**: Il ristorante a cui si riferisce la recensione.
- **stelle (int)**: Il punteggio assegnato, su una scala da 1 a 5.
- **testo (String)**: Il testo descrittivo del commento.
- **risposta (String)**: La risposta data dal ristorante alla recensione. Inizialmente è impostato a null.

3. Metodi e Funzionalità

La classe Recensione fornisce metodi per la gestione dei dati incapsulati:

- **Costruttore**: Recensione (Utente utente, Ristorante ristorante, int etoiles, String texte): Inizializza un nuovo oggetto Recensione con i dati forniti e imposta la risposta a null.
- **Metodi Getter**: Sono disponibili metodi pubblici per accedere ai valori di ogni campo (**getRistorante()**, **getUtente()**, **getStelle()**, **getTesto()**, **getRisposta()**).

- **Metodi Setter:**
 - **setStelle(int etoiles):** Permette di aggiornare il punteggio della recensione, ma solo se il valore rientra nell'intervallo 1-5.
 - **setTesto(String texte):** Consente la modifica del testo della recensione.
 - **setRisposta(String repense):** Permette a un ristorante di rispondere alla recensione. La risposta può essere impostata una sola volta.
- **toString():** Questo metodo sovrascritto crea una stringa formattata che rappresenta la recensione. Include il punteggio, il testo, il nome utente e, se presente, anche la risposta del ristorante.

4. Relazioni e Progettazione

- **Relazione di Composizione:** La classe Recensione è strettamente dipendente da altre classi. Un oggetto Recensione contiene riferimenti a un oggetto Utente e a un oggetto Ristorante.
- **Ruolo nel sistema:** L'istanza di Recensione viene creata dalla classe Cliente tramite il metodo lasciareRecensionir(). Viene poi memorizzata nella mappa del cliente e nella lista delle recensioni del ristorante. La classe Ristoratore interagisce con la classe Recensione per aggiungere risposte.

Ristorante.java

1. Introduzione

La classe Ristorante è stata progettata per modellare un'entità "ristorante" all'interno di un'applicazione. Essa gestisce le informazioni fondamentali di un ristorante, come nome, posizione geografica, tipo di cucina e fascia di prezzo, e permette di associare a esso un elenco di recensioni.

2. Struttura e Campi

La classe contiene i seguenti campi privati che ne definiscono le proprietà:

- **Nome (String)**: Il nome del ristorante.
- **Paese (String)**: Il paese in cui si trova il ristorante.
- **Città (String)**: la città in cui si trova il ristorante.
- **Indirizzo(String)**: L'indirizzo completo del ristorante.
- **Latitudine(double)**: la coordinata di latitudine per la geolocalizzazione.
- **Longitudine(double)**: la coordinata di longitudine per la geolocalizzazione.
- **fasciadiprezzo(double)**: la fascia di prezzo del ristorante.
- **Delivery(boolean)**: indica se è possibile effettuare il servizio di consegna a domicilio.
- **Prenotazione(boolean)**: indica se è possibile effettuare una prenotazione online.
- **TipoCucina(String)**: la tipologia di cucina offerta.
- **recensione(list<Recensione>)**: Un elenco di recensione associate al ristorante, inizializzato come un `Arraylist`

3. Metodi e Funzionalità

3.1 Costruttore

public Ristorante (String nome, String paese, String citta, String indirizzo, double latitudine, double longitudine, double fasciadelprezzo, boolean delivery, boolean prenotazione, String tipocucina)

È il costruttore per creare una nuova istanza della classe Ristorante inizializzando tutti gli attribute fondamentali.

3.2 Metodi principali

- **public String getNome():** Restituisce il nome del ristorante.
- **public String getCitta():** Restituisce la città del ristorante.
- **public double getFasciadiprezzo():** Restituisce la fascia di prezzo del ristorante.
- **public String getTipoCucina():** Restituisce il tipo di cucina.
- **public List<Recensione>getRecensione():** Restituisce la lista di recensione associate al ristorante
- **public void aggiungeRecensione(Recensione r):** Aggiunge un oggetto Recensione alla lista delle recensione(avis).
- **public double getMediaStelle():** Calcola e restituisce la media delle stelle assegnate nelle recensioni.

NOTA: Se la lista delle recensioni è vuota, il metodo restituisce 0.0 per evitare la divisione per zero. Altrimenti, itera su tutte le recensioni, somma i valori delle stelle e divide il totale per il numero di recensioni.

- **@Override public String toString():** Sovrascrive il metodo toString() per fornire una rappresentazione testuale formattata dell'oggetto Ristorante.
- **Valore di ritorno:** Una stringa che include nome, città, paese, fascia di prezzo e la media delle stelle. Esempio: "La Pergola (Roma, Italia) - 3.5€ | Media: 4.5★"

4. Dipendenze

Richiede la classe Recensione per gestire le recensioni.

Importa anche le classi java.util.ArrayList e java.util.List per la gestione dell'elenco di recensioni.

Ricerca.java

1. Introduzione

La classe Ricerca è una classe di utilità che fornisce metodi statici per eseguire operazioni di ricerca all'interno di una collezione di oggetti Ristorante. La sua funzione principale è quella di filtrare i ristoranti in base a criteri specifici, come la città, in modo efficiente e normalizzato. Essendo una classe di utilità, non ha attributi di istanza e i suoi metodi sono accessibili direttamente dal nome della classe.

2. Struttura e campi

Non contiene attributi di istanza.

3. Metodi e Funzionalità

- **public static List<Ristorante> ricercaPerCitta(List<Ristorante> ristorante, String città):** Questo metodo statico filtra una lista di ristoranti e restituisce una nuova lista contenente solo quelli che si trovano nella città specificata.

4. La logica di funzionamento

1. Viene creata una nuova lista, `resultats`, che conterrà i ristoranti trovati.
2. La stringa città in input viene "normalizzata": vengono rimossi gli spazi iniziali e finali (`.trim()`) e viene convertita in minuscolo (`.toLowerCase()`). Questo garantisce che la ricerca sia insensibile alle maiuscole/minuscole e agli spazi extra.
3. Il metodo itera su ogni oggetto `Ristorante` nella lista `restaurants`.
4. Per ogni ristorante, la sua città (`r.getCitta()`) viene anch'essa normalizzata.
5. Viene eseguito un confronto esatto (`.equals()`) tra la città del ristorante normalizzata e la stringa di ricerca normalizzata.
6. Se i nomi delle città corrispondono, il ristorante viene aggiunto alla lista `resultats`.
7. Al termine del ciclo, la lista `resultats` viene restituita.

Valore di ritorno: Una `List<Ristorante>` contenente tutti i ristoranti che soddisfano il criterio di ricerca. Se nessun ristorante viene trovato, viene restituita una lista vuota.

5. Dipendenze

Richiede la classe `Ristorante` e le classi `java.util.List` e `java.util.ArrayList`.

TheKnife.java

1. Introduzione

La classe TheKnife è il punto di ingresso principale dell'applicazione. Contiene il metodo main, che è il primo a essere eseguito quando l'applicazione viene avviata. Il suo scopo è avviare il programma, visualizzare un messaggio di benvenuto e inizializzare il menu principale.

2. Struttura e campi

Non contiene attributi di istanza.

3. Metodi e Funzionalità

`public static void main(String[] args):` Questo è il metodo principale dell'applicazione. Viene eseguito automaticamente all'avvio del programma

4. La logica di funzionamento

- Stampa un messaggio di benvenuto sulla console: "Benvenuto/a in TheKnife!".
- Crea una nuova istanza della classe Menu.
- Chiama il metodo `visualizzareMenuPrincipal()` sull'oggetto Menu per mostrare il menu principale all'utente.

5. Dipendenze

Richiede la classe Menu.