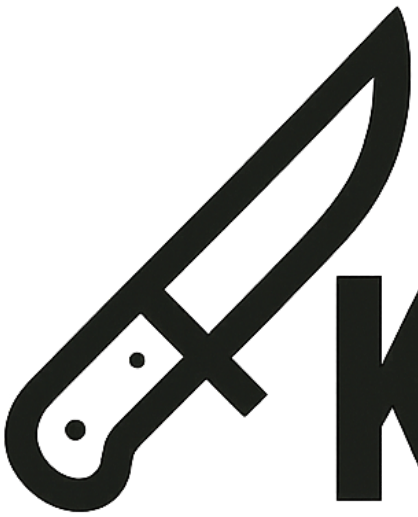

THE KNIFE

MANUALE TECNICO

- Scolaro Gabriele
- Ciani Flavio Angelo
- Gasparini Lorenzo

01/09/2025

Indice

Sommario

Introduzione

TheKnife è un'applicazione sviluppata come parte del **Laboratorio Interdisciplinare A** del corso di laurea in Informatica presso l'Università degli Studi dell'Insubria. Il progetto è scritto interamente in Java 24 ed è stato implementato e testato nei sistemi operativi Windows 10/11 e MacOS.

L'applicazione rappresenta un sistema solido e completo per la gestione di ristoranti, utenti, recensioni e ogni aspetto che li pone in relazione. L'architettura dell'applicazione è stata realizzata a strati, separando chiaramente la logica di presentazione, la logica dei servizi e la gestione dei dati, rendendo la struttura intuitiva e facilitandone l'estensione in futuro.

Librerie esterne usate

Per lo sviluppo dell'applicazione TheKnife sono state utilizzate diverse librerie di terze parti, in modo da gestire efficacemente funzionalità specifiche:

- **Apache HttpClient**

  Versione: httpclient-4.5.13.jar, httpcore-4.4.13.jar

  Scopo: gestione avanzata delle richieste HTTP.

  Utilità nel progetto: geolocalizzazione e recupero coordinate da indirizzi.

  Descrizione: supporta protocolli HTTP/1.1 e HTTP/2, gestione di connessioni persistenti, timeout, redirect, cookie e autenticazione; utilizzata per comunicare con API REST esterne (come servizi di geocoding).

- **JOpenCage**

  Versione: jopencage-1.4.0.jar

  Scopo: geocoding e reverse geocoding tramite il servizio **OpenCage Data**.

  Utilità nel progetto: geolocalizzazione e recupero coordinate da indirizzi.

  Descrizione: utilizzata per l'integrazione con il servizio di geocoding, fornisce un'API Java per convertire indirizzi in coordinate geografiche (latitudine/longitudine) e viceversa.

- **SLF4J (Simple Logging Facade for Java)**

  Versione: slf4j-api-1.7.30.jar, slf4j-simple-1.7.30.jar

  Scopo: astrazione per il logging.

  Utilità nel progetto: logging strutturato e diagnostica.

Descrizione: fornisce un'API unificata per il logging, permettendo di disaccoppiare la logica di registrazione dai dettagli dell'implementazione e facilitando la sostituzione del motore di logging senza modificare il codice.

- ## Apache Commons Logging

  Versione: commons-logging-1.2.jarAPI

  Scopo: API di logging compatibile con vari framework.

  Utilità nel progetto: logging strutturato e diagnostica.

  Descrizione: offre un'astrazione per il logging, spesso usata in contesti legacy o in combinazione con altre librerie Apache.

- ## Apache Commons Lang

  Versione: commons-lang3-3.12.0.jar

  Scopo: fornire utility e metodi helper per le operazioni comuni sul core di Java.

  Utilità nel progetto: offre funzionalità aggiuntive per la manipolazione di stringhe, array, gestione di eccezioni, metodi toString () automatizzati, e altre utility che estendono le capacità della standard library di Java.

  Descrizione: la libreria include classi come StringUtils, ArrayUtils, ExceptionUtils, e SystemUtils che forniscono metodi null-safe e semplificano operazioni comuni che sarebbero verbose con le API standard di Java.

- ## OpenCSV

  Versione: opencsv-3.8.jar

  Scopo: lettura e scrittura di file CSV.

  Utilità nel progetto: persistenza dei dati utenti, ristoranti e recensioni in formato CSV.

  Descrizione: libreria semplice ed efficiente per parsing e generazione di CSV. Supporta mapping tramite annotazioni, gestione di valori quotati, separatori personalizzati e handling di campi vuoti.

- ## Java Standard Library

  Moduli utilizzati:

  1. java.security (MessageDigest per SHA-256)
  2. java.time (LocalDate per gestione date)
  3. java.util (collections, scanner, properties)

  Scopo: funzionalità core del linguaggio.

  Utilità nel progetto: funzionalità di base crittografiche (hashing password) e gestione date.

  Descrizione: fornisce le strutture dati fondamentali, utilità per I/O, sicurezza, gestione di date e altro senza dipendenze esterne.

- **Jackson**

  Versioni: jackson-core, jackson-annotations, jackson-databind (versione 2.13.0)

  Scopo: serializzazione e deserializzazione di dati in formato JSON.

  Utilità nel progetto: gestione di API REST esterne.
  Descrizione: libreria ad alte prestazioni per elaborare dati JSON. Supporta annotazioni per il mapping oggetto-JSON, streaming API per elaborazione efficiente e formattazione personalizzata.

Architettura del sistema

Il sistema è organizzato in quattro package principali:

- **theknife.csv**: Gestione della persistenza su file CSV
- **theknife.eccezioni**: Gestione delle eccezioni personalizzate
- **theknife.logica**: Gestione degli aspetti logistici e servizi applicativi
- **theknife.utente/ristorante/recensione**: Modelli del dominio

Struttura del sistema di classi

L'architettura del progetto TheKnife segue un modello a strati con una chiara separazione delle responsabilità. Il sistema è organizzato in package logici che riflettono il dominio dell'applicazione (ristoranti, utenti, recensioni) e gli aspetti tecnici (persistenza, aspetti logistici, eccezioni).

1. Classi di dominio

a. Utente

Rappresenta un utente del sistema dotato di attributi fondamentali legati alla sua identità (nome, username, password, ruolo…). Comprende metodi appositi per la gestione di un utente.

b. Ristorante

Rappresenta un oggetto di tipo ristorante dotato di attributi fondamentali (nome, posizione, prezzo, recensioni…). Comprende metodi appositi per la gestione di un ristorante.

c. Recensione

Rappresenta un oggetto di tipo recensione dotato di attributi fondamentali (commento, username autore, stelle…). Comprende metodi appositi per la gestione di una recensione.

d. Ruolo (enum)

Definisce il tipo di utente (cliente o ristoratore) e determina le operazioni consentite.

2. Classi di servizio

a. DataContext

Classe che centralizza l'accesso ai dati e mantiene lo stato dell'applicazione. Carica e salva utenti, ristoranti e recensioni da/verso i CSV e ricostruisce gli indici interni per la ricerca rapida, oltre a fornire metodi per aggiungere, rimuovere e ricercare oggetti, fungendo quindi da "database" dinamico.

b. UtenteService, RistoranteService, RecensioneService

Classi di servizio che coordinano tutte le operazioni e le funzionalità legate agli oggetti che gestiscono.

c. GeoService

Astrazione per il geocoding con caching e gestione di errori. Comprende funzionalità come la serializzazione della cache su disco, fallback multipli per API key e algoritmo di **Haversine** per calcolo di distanze (formula sferica).

d. GestoreDate

Permette di gestire la conversione e la formattazione delle date, utilizzando un formato specifico in tutto il sistema.

e. Util

Comprende metodi di utilità generale (come hashing password SHA-256).

3. Classi per la gestione della persistenza

a. GestoreCSV (classe astratta)

Definisce il contratto per la serializzazione/deserializzazione CSV.

b. GestoreUtenti, GestoreRistoranti, GestoreRecensioni

Implementano la logica di parsing specifica per ogni entità, utilizzano OpenCSV per gestire quote, separatori e escaping e mappano da/verso oggetti di dominio con conversione dei tipi.

4. Classe per la gestione delle eccezioni

a. InputAnnullatoException

Eccezione non controllata per gestire annullamenti da parte dell'utente durante l'esecuzione.

5. Classi per la gestione della presentazione

a. MenuHandler

Controller principale che gestisce il flusso utente e coordina i servizi, comprendente costrutti switch-case per menu testuali.

b. TheKnife

Include il metodo main () e permette l'avviamento dell'applicazione.

Scelte progettuali

Nello sviluppo di TheKnife, le decisioni architetturali e implementative sono state guidate da un bilanciamento consapevole tra efficienza computazionale, semplicità di implementazione e manutenibilità del codice. Le scelte algoritmiche hanno considerato il contesto d'uso dell'applicazione, la dimensione prevedibile dei dataset e la necessità di rendere il sistema comprensibile e modificabile. Di seguito, un approfondimento sui punti chiave.

Gestione dei dati e persistenza:

La persistenza dei dati è affidata a file CSV gestiti attraverso una gerarchia di classi specializzate che estendono la classe astratta *GestoreCSV<T>*. Questa scelta offre diversi vantaggi:

- Semplicità e trasparenza: i file CSV sono leggibili e modificabili sia dall'applicazione che da strumenti esterni.
- Basso overhead: nessun requisito di database server o configurazioni complesse.
- Portabilità: i dati sono facilmente trasferibili tra diversi ambienti.

Il sistema utilizza la libreria **OpenCSV** per la gestione avanzata dei file CSV, con supporto per quote characters e escaping automatico, separatori personalizzati e handling di valori nulli e vuoti.

La classe DataContext funge da **facade centralizzato** per l'accesso ai dati, mantenendo indici in memoria per ricerche efficienti (HashMap per username e ID ristorante), cache delle associazioni utente-ristorante e coordinamento delle operazioni di salvataggio.

Le operazioni di lettura/scrittura avvengono principalmente all'avvio e alla chiusura dell'applicazione, con complessità O(n) per il caricamento e O(m) per il salvataggio, dove n e m rappresentano rispettivamente il numero di record nei file.

Gestione degli errori ed eccezioni

Il sistema adotta una strategia di gestione errori articolata su più livelli:

- InputAnnullatoException: eccezione unchecked personalizzata per gestire gli annullamenti utente senza contaminare le signature dei metodi.
- Validazione proattiva: check di precondizioni nei costruttori e metodi setter (es. validazione intervallo 1-5 per le stelle delle recensioni).

- Logging strutturato attraverso SLF4J e Apache Commons Logging per diagnostica e debugging.
- Graceful degradation: il sistema continua a funzionare anche in assenza di servizi esterni (ad esempio, geocoding disabilitato se manca API key).

Gestione delle password e sicurezza

La sicurezza delle credenziali è implementata attraverso un sistema di hashing SHA-256 tramite la classe Util. La scelta algoritmica offre:

- One-way hashing: le password non sono mai memorizzate in chiaro.
- Resistenza alle collisioni: proprietà crittografiche solide dell'algoritmo SHA-256.
- Migrazione trasparente: il sistema supporta automaticamente la conversione da password in chiaro a password hashate durante il login.

L'implementazione include un meccanismo di riconoscimento automatico del formato della password attraverso l'espressione regolare ^[0-9a-f]{64}$, che identifica gli hash SHA-256 esadecimali.

Geocoding e servizi di localizzazione

Il servizio di geolocalizzazione, implementato nella classe GeoService, utilizza **JOpenCage Geocoder** con le seguenti ottimizzazioni:

- Caching persistente: i risultati di geocoding vengono serializzati su disco (geocache.ser) per ridurre le chiamate API.
- Gestione elegante degli errori: fallback graceful in caso di API key mancante o errori di rete.

- **Algoritmo di Haversine** per il calcolo delle distanze geografiche (usato nel metodo calcolaDistanza)
- Suggerimenti intelligenti: filtraggio dei risultati duplicati e ordinamento per specificità.

Ottimizzazioni delle prestazioni

Le principali ottimizzazioni implementate includono:

- Lazy loading: le coordinate geografiche vengono calcolate solo quando necessarie.
- Indicizzazione in memoria: HashMaps per accesso O(1) a utenti e ristoranti.
- Caching strategico: risultati di geocoding e associazioni utente-ristorante.

- Early termination: nei filtri di ricerca per interrompere l'elaborazione quando possibile.

Considerazioni progettuali

Alcune scelte riflettono compromessi consapevoli:

- Nel progetto vengono utilizzate sia strutture sequenziali sia strutture associative. Le entità principali come utenti, ristoranti e recensioni sono mantenute in liste, che consentono di avere una collezione ordinata e facilmente serializzabile sui file CSV. Parallelamente, per garantire ricerche rapide ed evitare costosi scorrimenti lineari, sono stati introdotti indici basati su mappe. In particolare, il sistema mantiene una mappa che associa lo username all'oggetto utente, una che associa l'identificativo del ristorante al corrispondente oggetto ristorante e una che raggruppa le recensioni per ristorante. Questa scelta combinata permette di avere la semplicità e la compatibilità delle liste per le operazioni globali e di persistenza, insieme all'efficienza delle mappe per le operazioni di lookup, con tempi di accesso pressoché costanti.
- Gestione concorrenza minima: adeguata per un'applicazione single-user con operazioni principalmente in lettura.
- Validation locale: la validazione avviene principalmente lato client (input utente) per semplicità.

Queste scelte progettuali collettivamente contribuiscono a un sistema robusto, mantenibile ed efficiente, adatto allo scopo educativo del progetto mentre fornisce una base solida per estensioni future.

Limiti della soluzione sviluppata

Il sistema TheKnife, sebbene robusto e funzionale per lo scopo didattico, presenta alcuni limiti intrinseci dovuti a scelte progettuali volte alla semplicità e alla portabilità:

- **Scalabilità**: la persistenza dei dati basata su file CSV e il caricamento completo in memoria non sono ottimali per grandi volumi di dati. Operazioni su dataset molto grandi potrebbero risultare lente.
- **Concorrenza**: l'applicazione non è thread-safe e non supporta l'accesso concorrente. È progettata per un uso single-user e non è adatta a ambienti multi-utente senza sostanziali modifiche architetturali.
- **Sicurezza**: sebbene le password vengano hashate con SHA-256, non è

implementata una gestione avanzata della sicurezza come salt aggiuntivo, crittografia end-to-end dei dati sensibili o protezione da attacchi di forza bruta.

- **Dipendenze esterne**: il geocoding dipende da servizi terzi (OpenCage Geocoder) e dalla disponibilità di una connessione Internet attiva. In assenza di API key o di rete, alcune funzionalità risultano disabilitate.
- **Validazione input**: sebbene siano presenti controlli di base, non è implementata una validazione completa di tutti gli input utente, con possibili rischi di inconsistenza dati in caso di inserimenti malformati.
- **Mancanza di backup e transazioni**: non è previsto un sistema di backup automatico dei dati né meccanismi per garantire l'integrità dei dati in caso di errori durante le operazioni di scrittura.

Analisi delle classi principali

1. Classe Utente

La classe Utente rappresenta l'entità fondamentale del sistema, modellando un generico utente dell'applicazione con tutte le sue proprietà e comportamenti. Funge da classe base per la gestione delle identità nel sistema, implementando la logica di gestione delle credenziali, validazione anagrafica e associazioni con i ristoranti.

Attributi

- **Dati anagrafici**: nome, cognome, username, domicilio.
- **Credenziali**: username e password (hashata o in chiaro per migrazione).
- **Data di nascita**: data (oggetto LocalDate per gestione type-safe).
- **Ruolo**: ruolo (enum Ruolo con valori CLIENTE o RISTORATORE).
- Associazioni:
  - ristorantiPreferiti (List<Ristorante>) per utenti CLIENTE
  - ristorantiGestiti (List<Ristorante>) per utenti RISTORATORE
- Supporto persistenza: **assocKeysRaw** (stringa per serializzazione CSV).

Costruttori

Utente (nome, cognome, username, password, domicilio, data, ruolo): costruttore principale che inizializza tutti i campi obbligatori e crea le liste vuote per le associazioni.

Metodi principali

- **Getter e setter**
  - Getter standard per tutti gli attributi principali.
  - Setter limitati ai campi modificabili (nome, cognome, password).
- **Metodi di supporto alla persistenza**
  - getAssocKeysRaw () / setAssocKeysRaw (String s): gestiscono la serializzazione delle associazioni per il salvataggio CSV.
- **Metodi standard**
  - toString (): restituisce una rappresentazione leggibile dell'utente.
  - equals (Object obj): confronto basato sullo username (case insensitive).
  - hashCode (): coerente con equals, basato sullo username in minuscolo.

- **Gestione delle associazioni**
  - aggiungiAssoc (Ristorante r): aggiunge un ristorante alla lista appropriata in base al ruolo, con controllo duplicati.
  - rimuoviAssoc (Ristorante r): rimuove un ristorante dalla lista appropriata.
  - visualizzaAssoc (): mostra le associazioni.
  - gestisce (Ristorante r): verifica se un ristoratore gestisce un ristorante specifico.

2. Classe Ristorante

La classe Ristorante rappresenta è centrale nel dominio dell'applicazione, modellando un ristorante con tutte le sue informazioni e proprietà. La classe gestisce sia i dati identificativi e descrittivi del ristorante, sia le recensioni associate, fornendo metodi appositi per la gestione delle valutazioni.

Attributi

- **Dati identificativi**
  - id: identificativo univoco generato automaticamente.
  - nome: nome del ristorante.
  - indirizzo: via e numero civico.
  - location: città o area geografica.
- **Dati descrittivi**
  - prezzoMedio: fascia di prezzo ("€€" o valore numerico).
  - cucina: tipologia di cucina offerta.
  - premi: riconoscimenti e premi ottenuti.
  - servizi: servizi specifici messi a disposizione.
- **Coordinate geografiche**
  - latitudine: coordinata latitudinale in gradi decimali.
  - longitudine: coordinata longitudinale in gradi decimali.
- **Contatti**
  - numeroTelefono: recapito telefonico.
  - websiteUrl: sito web del ristorante.
- **Servizi boolean**
  - prenotazioneOnline: flag per prenotazione online.
  - delivery: flag per consegna a domicilio.
- **Gestione recensioni**
  - listaRecensioni: arrayList contenente tutte le recensioni associate.

Costruttori

- Ristorante (…): costruttore completo che inizializza tutti i campi e genera automaticamente un ID univoco tramite generaIDUnivoco (). Inizializza la lista delle recensioni vuota.

Metodi Principali

- **Gestione recensioni**
  - aggiungiRecensione (Recensione recensione): aggiunge una recensione non nulla alla lista (complessità O(1)).
  - esisteRecensioneDiUtente (String username): verifica l'esistenza di recensioni per utente (complessità O(n)).
  - trovaRecensioniDiUtente (String username): restituisce la prima recensione di un utente (complessità O(n)).
  - mediaStelle (): calcola la media aritmetica dei voti (complessità O(n)).
- **Getter e setter**
  - Getter completi per tutti gli attributi
  - Setter per tutti i campi modificabili (eccetto listaRecensioni).
- **Metodi di utilità**
  - generaIDUnivoco (): genera ID univoco usando UUID (8 caratteri maiuscoli).
  - toString (): restituisce rappresentazione dettagliata in formato JSON-like.

3. Classe Recensione

La classe Recensione modella la valutazione di un utente verso un ristorante, gestendo il sistema di feedback dell'applicazione. La classe implementa un modello completo che include sia il giudizio dell'utente che l'eventuale risposta del ristoratore, con validazioni robuste e gestione temporale.

Attributi principali

- **Campi immutabili** (final)
  - username: identificativo dell'autore della recensione.
  - idRistorante: identificativo del ristorante recensito.
- **Campi mutabili**
  - stelle: valutazione in stelle (range 1-5, con validazione).
  - commento: testo della recensione (con handling di valori null).
  - data: data della recensione (oggetto LocalDate).
  - risposta: eventuale risposta del ristoratore (con handling di valori null).

Costruttori

- **Costruttore base**
  - Recensione (username, idRistorante, stelle, commento): crea una nuova recensione con data corrente e risposta vuota.
- **Costruttore completo**
  - Recensione (username, idRistorante, stelle, commento, data, risposta): permette di specificare tutti i campi.

Metodi principali

- **Getter e setter**
  - Getter per tutti i campi con naming semantico (getAutore () invece di getUsername ()).
  - Setter con validazione: setStelle () valida il range 1-5.
  - Setter con null-safety: setDescrizione () e setRisposta () gestiscono valori null.
- **Validazione**
  - checkStelle (int v): metodo privato statico che valida il range delle stelle (1-5); solleva IllegalArgumentException per valori non validi.
- **Operazioni utili**
  - visualizzaRecensione (): restituisce una rappresentazione formattata per l'utente.
  - isPositiva (): determina se la recensione è positiva (4+ stelle).
  - isRecente (): verifica se la recensione è entro 30 giorni.
  - modificaRecensione (): aggiorna stelle e commento, reimpostando la data a oggi.

    - eliminaRisposta (): rimuove la risposta del ristoratore.
- **Metodi standard**
    - toString (): rappresentazione completa per debugging.
    - equals () e hashCode (): basati su autore, ristorante, data, commento e stelle.

4. Classe UtenteService

La classe UtenteService si occupa della gestione degli utenti, incapsulando la logica applicativa per l'autenticazione, registrazione e gestione delle associazioni utente-ristorante. Funge da facade tra il layer di presentazione (MenuHandler) e il data layer (DataContext), implementando le regole di business e la sicurezza.

Attributi

- data: riferimento al DataContext (iniezione dipendenza via costruttore). Final e obbligatorio, garantisce che il servizio sia sempre correttamente inizializzato.

Costruttori

- UtenteService (DataContext data): costruttore unico che richiede l'istanza di DataContext. Implementa il pattern dependency injection e garantisce che il servizio abbia sempre accesso ai dati.

Metodi

- **Gestione autenticazione**
  - registrazione (Utente nuovo): registra un nuovo utente hashandone la password.
  - login (String username, String passwordPlain): autentica un utente con supporto migrazione password.
  - isSha256Hex (String s): verifica il formato hash SHA-256.
- **Gestione associazioni**
  - visualizzaPreferiti (String username): visualizza i ristoranti preferiti di un cliente.
  - aggiungiRistoranteGestito (String username, Ristorante r): assegna un ristorante a un ristoratore con controllo duplicati.
  - rimuoviRistoranteGestito (String username, Ristorante r): rimuove un ristorante dalla gestione.
- **Metodi di ricerca**
  - trovaUtente (String username): delegata al DataContext per ricerca utente.

5. Classe RistoranteService

La classe RistoranteService si occupa della gestione completa dei ristoranti, fornendo servizi per tutte le operazioni legate alla ricerca, filtro e gestione dei ristoranti; Supporta tre diverse categorie di utenti: ospiti, clienti e ristoratori, con funzionalità appropriate per ciascun ruolo.

Attributi

- dataContext: riferimento al DataContext per l'accesso ai dati.
- geoService: riferimento al GeoService per le operazioni di geolocalizzazione.

Entrambi i campi sono finali e obbligatori nel costruttore.

Costruttori

- RistoranteService (DataContext dataContext, GeoService geoService): costruttore con dependency injection; riceve tutte le dipendenze necessarie per il funzionamento e garantisce che il servizio sia completamente inizializzato.

Metodi

- **Ricerca e filtri** (per tutti gli utenti)
  - cercaRistorantePerFiltri (): ricerca avanzata con multiple condizioni di filtro.
  - matchesCriteri (): verifica se un ristorante soddisfa i criteri di ricerca.
  - matchesFasciaPrezzo (): converte e confronta le fasce di prezzo.
- **Gestione preferiti** (per i clienti)
  - aggiungiPreferito (): aggiunge un ristorante ai preferiti di un cliente.
  - rimuoviPreferito (): rimuove un ristorante dai preferiti.
- **Gestione ristoranti** (per i ristoratori)
  - aggiungiRistorante (): aggiunge un nuovo ristorante al sistema.
- **Geolocalizzazione** (per tutti)
  - cercaVicinoA (): ricerca ristoranti entro una certa distanza geografica.

6. Classe RecensioneService

La classe RecensioneService si occupa della gestione completa del ciclo di vita delle recensioni, implementando le regole di dominio e i controlli di sicurezza per tutte le operazioni relative alle valutazioni degli utenti; garantisce l'integrità del sistema di recensioni attraverso validazioni rigorose e controlli di autorizzazione.

Attributi

- dataContext: riferimento al DataContext (dependency injection via costruttore). Final e obbligatorio, garantisce l'accesso consistente ai dati ed è utilizzato per operazioni di persistenza delle recensioni.

Costruttori

- RecensioneService(DataContext dataContext): costruttore unico con dependency injection; riceve il DataContext come dipendenza obbligatoria e implementa il principio di inversione del controllo.

Metodi

- **Gestione recensioni utente**
  - aggiungiRecensione (): crea una nuova recensione con controlli di duplicazione.
  - modificaRecensione (): permette la modifica di recensioni esistenti.
- **Gestione risposta ristoratore**
  - rispondiRecensione (): permette ai ristoratori di rispondere alle recensioni.

7. Classe GeoService

La classe GeoService rappresenta il servizio di geolocalizzazione dell'applicazione, fornendo funzionalità avanzate di geocoding e calcolo distanze; integra l'API esterna JOpenCage Geocoder con un sistema di caching sofisticato e implementa algoritmi geospaziali per le ricerche basate sulla posizione.

Attributi

- apiKey: chiave API per JOpenCage (final, inizializzata nel costruttore).
- geocodeCache: HashMap per cache locale indirizzo -> coordinate.
- CACHE_FILE: path costante per la persistenza della cache su filesystem.

Costruttori

- GeoService (): costruttore che inizializza la API key e carica la cache da disco; chiama loadApiKey () per il recupero delle credenziali e chiama loadCacheFromFile () per il ripristino della cache.

Metodi

- **Gestione API Key**
  - loadApiKey (): carica la chiave API con fallback multipli (file -> variabile d'ambiente).
- **Gestione cache**
  - loadCacheFromFile (): deserializza la cache da filesystem.
  - saveCacheToFile (): Serializza la cache su filesystem.
  - shutdown (): metodo pubblico per salvare la cache all'uscita.
  - clearCache (), getCacheSize (): metodi di utilità per la gestione cache.
- **Geocoding**
  - geocode (String indirizzo): converte indirizzi in coordinate con caching.
- **Ricerca geospaziale**
  - filtraPerVicinoA (): filtra ristoranti per distanza geografica.
  - calcolaDistanza (): implementa l'algoritmo di Haversine per calcolo distanze.

8. Classe DataContext

La classe DataContext rappresenta il cuore del sistema di gestione dati dell'applicazione, funzionando come **unità di lavoro** centralizzata che coordina tutte le operazioni sui dati. Implementa un pattern **Repository** avanzato che combina persistenza CSV con indici in memoria per performance ottimali, gestendo la consistenza tra le tre entità principali: utenti, ristoranti e recensioni.

Architettura a strati

- **Layer di persistenza**
  - GestoreUtenti, GestoreRistoranti, GestoreRecensioni: gestori specializzati per la serializzazione CSV.
  - Separazione delle responsabilità tra formattazione CSV e business logic.
- **In-Memory layer**
  - Liste complete di tutte le entità caricate.
  - Indici per accesso rapido alle entità.
- **Layer di indicizzazione**
  - Mappe per ricerca efficiente O(1) tramite chiavi primarie.

Struttura dati principale

- **Collezioni primarie**
  - Private List<Utente> utenti;
  - Private List <Ristorante> ristoranti;
  - Private List <Recensione> recensioni;
- **Indici di ricerca**
  - private Map<String, Utente> utentiPerUsername;
    O(1) per username.
  - private Map<String, Ristorante> ristorantePerId;
    O(1) per ID.
  - private Map<String, List<Recensione> recensioniPerRistoId;
    O(1) per accesso e O(k) per iterazione.

Metodi

- **Operazioni di bulk**
  - loadAll (): caricamento completo di tutti i dati con rebuilding indici.
  - saveAll (): salvataggio di tutti i dati su CSV.

- Operazioni CRUD
  - addUtente (), addRistorante (), addRecensione (): inserimenti con consistenza automatica.
  - removeRecensione (): rimozione con cleanup completo degli indici.
  - findUtente (), findRistoranteById (): ricerche efficienti O(1).
- Gestione relazioni
  - collegaRecensioniAiRistoranti (): collegamento bidirezionale recensioni <-> ristoranti.
  - linkAssocUtenti (): ricostruzione associazioni utente-ristoranti da CSV.
  - buildAssocKeysRaw (): serializzazione associazioni per persistenza.

9. Classe GestoreUtenti

La classe GestoreUtenti è una specializzazione concreta della classe astratta GestoreCSV<T> dedicata alla gestione della persistenza degli oggetti Utente su file CSV. Implementa le operazioni specifiche di parsing e serializzazione per il formato degli utenti, gestendo la conversione tra oggetti Java e rappresentazione tabulare CSV.

Ereditarietà e implementazione

- Estende GestoreCSV<Utente> ereditando il campo elementi (List<Utente>) e il contratto dei metodi astratti.
- Implementa i metodi caricaDaCSV () e salvaSuCSV () con la logica specifica per gli utenti.

Struttura del file CSV

Il file segue il formato:
Nome,Cognome,Username,Password,Domicilio,Data,Ruolo,Associati

Metodi

- caricaDaCSV (String filePath)
  Implementa il caricamento degli utenti dal file CSV con le seguenti caratteristiche:
  - **Flusso di caricamento**:
    - Ignora intestazione: salta la prima riga (header).
    - Validazione minima: richiede almeno 7 colonne per riga.
    - Parsing campi: trim degli spazi e gestione valori null.
    - Conversione dati:
      - Conversione stringa -> LocalDate tramite GestoreDate.parseNullable ().
      - Conversione stringa -> Ruolo tramite Ruolo.valueOf ().
    - Gestione associazioni: caricamento della stringa raw delle associazioni.
  - **Gestione errori**:
    - Errori I/O: logging con messaggio descrittivo.
    - Errori parsing: logging con continuazione del processing.
    - Righe malformate: skip con messaggio di warning.
- salvaSuCSV (String filePath)
  Implementa il salvataggio degli utenti su file CSV con:
  - Flusso di salvataggio:
    - Scrittura intestazione: header con nomi colonne.
    - Serializzazione campi: conversione oggetti -> stringhe CSV.
    - Formattazione dati:

        - Data formattata con GestoreDate.formatOrEmpty ().
        - Ruolo convertito con Ruolo.name ().
    - Associazioni: serializzazione della stringa raw.
- **Gestione errori**:
    - Errori I/O: logging con messaggio descrittivo.
    - Errori di serializzazione: gestiti implicitamente da OpenCSV.

10. Classe GestoreRistoranti

La classe GestoreRistoranti è una specializzazione concreta della classe astratta GestoreCSV<Ristorante> dedicata alla gestione della persistenza degli oggetti Ristorante su file CSV. Implementa operazioni avanzate di parsing e serializzazione per il formato dei ristoranti, gestendo la conversione di diversi tipi (stringhe, numeri, booleani) tra oggetti Java e rappresentazione CSV.

Ereditarietà e implementazione

- Estende GestoreCSV<Ristorante> ereditando il campo elementi (List<Ristorante>) e il contratto dei metodi astratti.
- Implementa i metodi caricaDaCSV () e salvaSuCSV () con logica specifica per i ristoranti.

Struttura del file CSV

Il file segue un formato complesso con 14 colonne:
ID,Nome,Indirizzo,Location,Prezzo,Cucina,Longitudine,Latitudine,Telefono,Sito Web,Premio,Servizi,PrenotazioneOnline,Delivery

Metodi principali

- caricaDaCSV (String filePath)
  Implementa il caricamento dei ristoranti con parsing avanzato:
  - **Flusso di caricamento**:
    - Parser Configurato: utilizza CSVParserBuilder con separatore personalizzato.
    - Validazione strutturale: richiede 14 colonne complete per riga.
    - Conversione tipi complessi:
      - Coordinate geografiche: stringa -> double con gestione separatori.
      - Valori booleani: stringa -> boolean con supporto multilingua.
      - ID obbligatorio: validazione presenza identificativo.
  - **Gestione errori avanzata**:
    - Errori di formato: skip riga con messaggio diagnostico.
    - Errori di conversione: valori di default con logging.
    - Dati mancanti: gestione con valori di fallback.
- salvaSuCSV (String filePath)
  Implementa il salvataggio con formattazione consistente:
  - **Flusso di salvataggio**:
    - Intestazione dettagliata: 14 colonne con naming esplicito.
    - Formattazione specifica:
      - Booleani convertiti in "SI"/"NO" (localizzato).
      - Numeri formattati con separatore punto.

- Stringhe protette con quoting automatico.

Metodi di supporto

- parseDouble (String valore): gestisce valori null e vuoti, supporta sia punto che virgola come separatore e restituisce 0.0 per errori (fallback).
- parseBool (String valore): dotato di supporto multilingua, restituisce false per valori non riconosciuti.

11. Classe GestoreRecensioni

La classe GestoreRecensioni è una specializzazione concreta della classe astratta GestoreCSV<Recensione> dedicata alla gestione della persistenza degli oggetti Recensione su file CSV. Implementa operazioni specifiche di parsing e serializzazione per il formato delle recensioni, gestendo la struttura dati che include metadati temporali, valutazioni numeriche e testo libero.

Ereditarietà e implementazione

- Estende GestoreCSV<Recensione> ereditando il campo elementi (List<Recensione>) e il contratto dei metodi astratti.
- Implementa i metodi caricaDaCSV () e salvaSuCSV () con logica specifica per le recensioni.

Struttura del file CSV

Il file segue il formato:
Username,IDRistorante,Stelle,Commento,Data,Risposta

Metodi

- caricaDaCSV (String filePath)
  Implementa il caricamento delle recensioni con gestione avanzata dei casi limite:
  - **Flusso di caricamento**:
    - Validazione minima: richiede almeno 4 colonne fondamentali.
    - Gestione valori null: conversione esplicita di null in stringhe vuote.
    - Conversione dati complessi:
      - Stelle: conversione stringa -> int con validazione range implicita.
      - Date: parsing con GestoreDate.parseNullable () per gestione valori mancanti.
      - Campi opzionali: gestione colonne aggiuntive per risposta e data.
  - **Gestione errori avanzata**:
    - Errori formato stelle: skip riga con messaggio diagnostico.
    - Errori parsing: logging dettagliato con continuazione.
    - Dati incompleti: utilizzo valori default per campi opzionali.
- salvaSuCSV (String filePath)
  Implementa il salvataggio con formattazione ottimizzata:
  - Flusso di salvataggio:
    - Intestazione chiara: 6 colonne con naming consistente.

- Formattazione specifica:
    - Date formattate con pattern coerente "dd/MM/yyyy".
    - Valori numerici convertiti esplicitamente.
    - Testo protetto con quoting automatico.

12. Classe MenuHandler

La classe MenuHandler rappresenta il layer di presentazione dell'applicazione TheKnife, fungendo da controller principale che orchestra tutti i flussi utente attraverso un'interfaccia a riga di comando; coordina servizi e output console, gestendo l'intero ciclo di vita dell'applicazione.

Architettura e responsabilità

- **Controller centrale**
  - Coordina tutti i servizi dell'applicazione.
  - Controlla la navigazione tra i diversi menu.
  - Gestisce l'interazione completa con l'utente.
- **Dependency management**
  - Utilizza e integra tutti i servizi principali (Utente, Ristorante, Recensione, Geo).
  - Mantiene il riferimento al DataContext per l'accesso ai dati.
  - Gestisce apertura/chiusura delle risorse (Scanner, GeoService).

Struttura dei menu

- **Menu principale** (4 opzioni)
  - Registrazione nuovo utente.
  - Login utente esistente.
  - Modalità ospite.
  - Uscita dall'applicazione.
- **Menu ospite** (3 opzioni)
  - Ricerca ristoranti con filtri.
  - Ricerca geografica.
  - Ritorno al menu principale.
- **Menu cliente** (5 opzioni)
  - Ricerca con filtri.
  - Ricerca "Vicino a Me"
  - Ricerca geografica.
  - Gestione preferiti.
  - Gestione recensioni.
  - Logout.
- **Menu ristoratore** (5 opzioni)
  - Creazione nuovo ristorante.
  - Gestione ristoranti esistenti.
  - Assunzione gestione ristoranti.
  - Gestione risposte recensioni.
  - Logout.

Pattern e strategie implementate

- **State pattern**
  - Gestione di stati differenti in base al ruolo utente (ospite/cliente/ristoratore).
  - Transizioni di stato controllate attraverso menu gerarchici.
- **Command pattern**
  - Ogni opzione di menu mappa a un comando specifico.
  - Separazione tra la selezione dell'opzione e l'esecuzione del comando.
- **Gestione delle eccezioni**
  - InputAnnullatoException per gestire le cancellazioni da parte dell'utente.
  - Gestione graceful degli errori con recovery.
- **Template method**
  - Flussi predefiniti per operazioni complesse (registrazione, ricerca, etc.).
  - Riutilizzo di codice attraverso metodi helper.

Componenti principali

- **Gestione input**
  - Molteplici metodi helper per la lettura validata dell'input.
  - Supporto per annullamento operazioni con carattere "*".
  - Validazione in real-time con messaggi di errore contestuali.
- **Gestione output**
  - Formattazione consistente dell'output.
  - Visualizzazione strutturata di liste e dettagli.
  - Supporto per pulizia schermo e pausa visualizzazione.

Flussi complessi implementati

- **Registrazione utente**
  - Validazione campi obbligatori.
  - Check username univoco.
  - Conferma password.
  - Scelta ruolo con validazione.
- **Ricerca avanzata**
  - Filtri multipli combinabili.
  - Ricerca geografica con geocoding.
  - Paginazione risultati.
  - Selezione interattiva.
- **Gestione recensioni**
  - Sistema di modifica e risposta.

    - Validazione autorizzazione.
    - Aggiornamento in tempo reale.
- **Creazione/Gestione ristorante**
    - Inserimento dati completi.
    - Geocoding automatico con conferma.
    - Validazione coordinate.
    - Modifica campi ristorante
    - Eliminazione ristorante

In sintesi

I modelli adottati nel progetto seguono un'architettura a strati, con una chiara separazione delle responsabilità. Lo strato della presentazione è affidato alla classe MenuHandler, che gestisce l'interazione con l'utente e il flusso dei menu. Al di sotto si colloca lo strato dei servizi, che comprende UtenteService, RistoranteService e RecensioneService: questi componenti incapsulano la logica di business e costituiscono il punto di accesso alle funzionalità applicative. Infine, lo strato della gestione dei dati è rappresentato dal DataContext e dai gestori CSV, responsabili del caricamento, salvataggio e collegamento delle entità.

Per quanto riguarda la gestione della logica applicativa, si adotta un modello basato sui servizi, così da isolare le operazioni principali dal codice di presentazione e garantire una maggiore manutenibilità. Inoltre, è stato definito un modello delle eccezioni: in particolare l'eccezione personalizzata InputAnnullatoException consente di gestire in modo chiaro e uniforme i casi in cui l'utente interrompe volontariamente un'operazione.

Sitografia

- JOpenCage Geocoder: https://opencagedata.com/api
- OpenCSV: http://opencsv.sourceforge.net/
- Apache Commons Lang: https://mvnrepository.com/artifact/org.apache.commons/commons-lang3/3.12.0
- SLF4J: http://www.slf4j.org/
- Apache Commons Logging: https://commons.apache.org/proper/commons-logging/
- Jackson JSON Processor: https://github.com/FasterXML/jackson
- Apache HttpClient: https://hc.apache.org/httpcomponents-client-4.5.x/

- Java Platform, Standard Edition Documentation: https://docs.oracle.com/javase/8/docs/api/

Conclusioni

Le scelte architetturali prese garantiscono una buona manutenibilità e estensibilità. Le principali aree di miglioramento futuro potrebbero includere:

- Migrazione a un database relazionale.
- Aggiunta di test automatizzati.