THE KNIFE

Manuale Tecnico

Sviluppatori: Marco Barlera & Semenzato Francesco 23/06/2025

Indice

Sommario

Introduzione

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

L'obiettivo principale è realizzare un sistema solido per la gestione di ristoranti e utenti, con un focus particolare sulla struttura a oggetti, l'integrità dei dati, l'interazione con servizi esterni (come il geocoding) e l'estensibilità futura. L'assenza di un'interfaccia grafica è una scelta deliberata, finalizzata a concentrare lo sviluppo sulla logica di dominio e sulla qualità architetturale del codice.

Librerie esterne usate

Per lo sviluppo del progetto TheKnife sono state adottate diverse librerie di terze parti, ciascuna scelta per gestire in modo efficiente funzionalità specifiche:

- ### Apache HttpClient
  Versione: httpclient-4.5.13.jar, httpcore-4.4.13.jar
  Utilizzata per gestire operazioni di rete e chiamate HTTP. Fornisce un'API affidabile e flessibile per la comunicazione con servizi web esterni.

- ### JOpenCage
  Versione: jopencage-1.4.0.jarLibreria utilizzata per l'integrazione con il servizio di geocoding OpenCage. Consente la conversione bidirezionale tra indirizzi e coordinate geografiche, fondamentale per la localizzazione dei ristoranti.

- ### SLF4J (Simple Logging Facade for Java)
  Versione: slf4j-api-1.7.30.jar, slf4j-simple-1.7.30.jar
  Fornisce un'astrazione per il logging. Permette di separare la logica di registrazione dai dettagli dell'implementazione, facilitando la sostituzione del backend di logging senza modifiche al codice.

- ### Apache Commons Logging
  Versione: commons-logging-1.2.jarAPI di logging generica, compatibile con vari framework di logging. Garantisce interoperabilità tra componenti anche in contesti ibridi o legacy.

- **Jackson**

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

  Framework per la serializzazione e deserializzazione di dati in formato JSON. È stato impiegato per elaborare e convertire i dati scambiati tramite API REST.

Struttura del sistema di classi

Il progetto è basato su un modello a oggetti, incentrato sulla gestione di ristoranti e utenti. L'architettura promuove la separazione delle responsabilità e l'organizzazione del codice. Le classi principali si suddividono in quattro gruppi funzionali:

1. ## Classi principali
    a. **Utente (classe astratta)**
    Definisce le proprietà e il comportamento base di ogni utente del sistema. Contiene la logica di gestione dell'identità, cifratura delle password e validazione anagrafica. È estesa da:
        - Cliente
        - Ristoratore
        - UtenteNonRegistrato

    b. **Ristorante**
    Rappresenta l'entità ristorante con attributi fondamentali (nome, posizione, prezzo, recensioni, ecc.).

    c. **Recensione**
    Modella il feedback dell'utente verso un ristorante. Contiene voto, commento, risposta e metodi di confronto.

2. ## Classi secondarie
    a. **Indirizzo, DataDiNascita**
    Classi utilizzate per la gestire alcuni dati dell'utente che richiedevano una implementazione distaccata rispetto ai parametri della classe Utente.

    b. **GestoreRistoranti**
    Classe di servizio che coordina tutte le operazioni legate ai ristoranti:
        - caricamento/salvataggio da file CSV
        - filtri per città, distanza, fascia di prezzo e tipo di cucina
        - geocoding(tramite JOpenCage)
        - statistiche aggregate

3. ## Classe per gestione eccezioni
    InputAnnullatoException: Eccezione personalizzata per gestire annullamenti da parte dell'utente nel flusso di esecuzione.

4. ## Classe TheKnife
    Classe che gestisce l'orchestrazione tra classi.
    Include il metodo main() e coordina le operazioni base di caricamento dati, interazione utente e salvataggio.

Scelte progettuali

Nello sviluppo del progetto, particolare attenzione è stata dedicata alla selezione degli algoritmi e delle strutture dati più adeguati, con l'obiettivo di ottenere un bilanciamento tra efficienza computazionale, semplicità di implementazione e manutenibilità del codice. La scelta algoritmica, infatti, non si è limitata alla mera ricerca della massima performance, ma ha considerato il contesto d'uso dell'applicazione, la prevedibile dimensione dei dataset e la necessità di rendere il sistema facilmente comprensibile e modificabile. Di seguito sono descritte le principali decisioni algoritmiche adottate, organizzate per area funzionale.

Gestione dei Dati e Ricerca:

La gestione dell'elenco dei ristoranti è affidata alla classe GestoreRistoranti, che utilizza strutture dati lineari, in particolare ArrayList, per la memorizzazione. Questa scelta è motivata dal fatto che le operazioni di inserimento e rimozione, frequenti in fase di inizializzazione o modifica del database, hanno complessità ammortizzata O(1). La ricerca di un ristorante per nome, realizzata tramite scansione lineare (getRistorante(nome)), ha complessità O(n). Tuttavia, questo costo è considerato trascurabile in un contesto con un numero relativamente limitato di elementi (decine o centinaia), come previsto per l'applicazione.

Le funzionalità di filtro avanzato (es. per città, categoria, fascia di prezzo, valutazione) sono implementate mediante catene di filtri in memoria. Sebbene la complessità cumulativa possa arrivare a O(n) per ciascun filtro, le condizioni vengono applicate in modo pigro (lazy), ottimizzando l'esecuzione in base all'ordine dei filtri e all'early termination.

Un'attenzione particolare è stata dedicata alle funzioni di geolocalizzazione. Il servizio di geocoding fornito da JOpenCage viene invocato solo alla prima richiesta per una determinata località, e il risultato viene memorizzato in cache. Questo approccio consente di ridurre drasticamente i tempi di risposta e il numero di richieste API, migliorando sia le performance che la sostenibilità del sistema. Inoltre, il calcolo delle distanze tra coordinate geografiche sfrutta la formula di Haversine, con tempo di esecuzione O(1) per ogni confronto.

Operazioni sulle Recensioni

Le recensioni associate a ciascun ristorante sono gestite anch'esse tramite ArrayList, in linea con l'approccio semplice ed efficiente scelto per l'intero sistema. L'aggiunta di una nuova recensione avviene in tempo O(1), mentre la ricerca o la rimozione (ad esempio nel metodo rimuoviRecensione) richiede una scansione completa della lista, con costo O(n). Questa complessità è ritenuta accettabile data la quantità contenuta di recensioni per ristorante. Il calcolo della media dei voti, necessario per classificare e ordinare i ristoranti, viene effettuato in tempo O(k), dove k è il numero di recensioni. Per migliorare le prestazioni, si evita il ricalcolo della media ad ogni modifica, adottando una strategia di aggiornamento incrementale o differito in base al contesto d'uso.

Crittografia delle Password

La protezione delle credenziali utente è affidata a un algoritmo di cifratura personalizzato, implementato nella classe Utente. Il processo, con complessità O(m) (dove m è la lunghezza della password), prevede uno shift deterministico dei caratteri ASCII per vocali e consonanti, oltre a trasformazioni modulari sui numeri.
Questa soluzione non ha ambizioni di sicurezza avanzata, ma è stata progettata con l'intento di fornire un livello base di offuscamento, sufficiente per un'applicazione didattica. L'adozione di un algoritmo proprietario consente di evitare l'inclusione di librerie esterne, semplificando la distribuzione del progetto e rafforzando l'indipendenza dal contesto di esecuzione.

Ottimizzazioni Trasversali

Oltre agli aspetti specifici delle singole funzionalità, sono state adottate alcune scelte architetturali orientate all'efficienza globale del sistema:
- Geocoding Lazy: le coordinate geografiche vengono calcolate e normalizzate solo nel momento in cui sono effettivamente necessarie, ad esempio per confronti di distanza o ordinamenti spaziali.
- Caricamento iniziale dei dati: i file CSV contenenti ristoranti, utenti e recensioni vengono letti una sola volta all'avvio dell'applicazione, con complessità *O(n) per ciascun file. I dati sono successivamente mantenuti in memoria, evitando accessi ripetuti al disco.
- Unione di risultati da filtri multipli: per evitare duplicati in presenza di più criteri di selezione combinati, i risultati vengono temporaneamente convertiti

in un LinkedHashSet, consentendo un'unione efficiente in O(n + m) e preservando l'ordine di inserimento.

Considerazioni progettuali

Alcune scelte sono state guidate da un compromesso consapevole tra prestazioni teoriche e semplicità pratica. Ad esempio:
- ArrayList vs HashMap: Sebbene le ricerche in HashMap offrano tempi costanti *O(1), la semplicità e la leggibilità delle ArrayList, unite al basso volume di dati, rendono quest'ultime preferibili per un progetto educativo.
- Assenza di indicizzazione strutturata: Non è stato adottato un database relazionale o indicizzato, per mantenere l'indipendenza del progetto da tecnologie esterne e facilitarne la portabilità. Tutti i filtri e le ricerche vengono eseguiti in memoria.
- Crittografia "leggera": La soluzione implementata non protegge da attacchi reali, ma è sufficiente per simulare un meccanismo di autenticazione in un contesto di sviluppo e testing.

Analisi delle classi principali

1. Classe Utente

Classe astratta che rappresenta un generico utente dell'applicazione (es. Cliente, Ristoratore, Guest).

Attributi principali:

- Nome: nome proprio dell'utente, con iniziale maiuscola.
- Cognome: cognome dell'utente, con iniziale maiuscola.
- Username: nome utente univoco per l'accesso.
- Password: password cifrata se utente nuovo, altrimenti salvata già cifrata.
- Domicilio: indirizzo di residenza.
- Ruolo: tipo di utente (Cliente, Ristoratore, Guest...).
- Data: oggetto DataDiNascita contenente la data di nascita.
- nuovo: flag booleano che indica se l'utente è appena registrato.

Costruttori

- Utente(): costruttore vuoto.
- Utente(...): due versioni:
  - con giorno, mese, anno come interi;
  - con data in formato stringa (gg/mm/aaaa o gg-mm-aaaa).

Metodi principali

- Getter e Setter per ogni attributo.
- CifraPassword(String): cifra la password seguendo queste regole:
  - vocali: +3 ASCII;
  - consonanti: +7;
  - cifre pari: -3 mod 10;
  - cifre dispari: -7 mod 10;
  - altri simboli: +1.
- DecifraPassword(String): applica l'inverso della cifratura.
- getPasswordDecifrata(String, String): restituisce la password in chiaro se quella inserita è corretta.
- toString(): restituisce una stringa con le informazioni principali dell'utente.

2. Classe cliente

Classe che rappresenta un utente registrato al sistema con ruolo Cliente.
Un cliente può:

- aggiungere o rimuovere ristoranti dai preferiti
- scrivere, modificare o eliminare recensioni

Estende la classe Utente.

Attributi principali:

- ListaRecensioniUtente → contiene tutte le recensioni scritte dal cliente
- ListaPreferiti → contiene tutti i ristoranti preferiti dal cliente
- Tutti i parametri della classe Utente.
- Ruolo → “Cliente”

Costruttori:

- Cliente(): costruttore vuoto
- Cliente(nome, cognome, username, password, domicilio, giorno, mese, anno, nuovo): inizializza il cliente con data separata
- Cliente(nome, cognome, username, password, domicilio, dataStringa, nuovo): inizializza il cliente con la data in formato stringa (es. "01/01/2000") → carica automaticamente le recensioni dal file

Metodi principali:

- Getter
  - getListaRecensioni(): restituisce la lista delle recensioni dell’utente
  - getPreferiti(): restituisce la lista dei ristoranti preferiti
  - getPreferitiString(): restituisce una stringa con i nomi dei ristoranti preferiti separati da _ (es. "McDonalds_BurgerKing")
- Metodi per caricare dati
  - CaricaListaRecensione(ArrayList rec): carica le recensioni dell’utente dal file CSV
  - CaricaListaPreferiti(username, gestoreRistoranti): carica i ristoranti preferiti leggendo il file utenti e usando il gestore ristoranti per recuperarli
- Metodi per i preferiti
  - AggiungiAiPreferiti(ristorante): aggiunge un ristorante ai preferiti (solo se non è già presente)
  - RimuoviPreferiti(ristorante): rimuove un ristorante dai preferiti

    - VisualizzaPreferiti(): restituisce una stringa con l’elenco dei preferiti
- Metodi per le recensioni
    - AggiungiRecensione(voto, commento, ristorante): crea una nuova recensione, la aggiunge alla lista dell’utente e a quella del ristorante
    - VisualizzaRecensioni(): stampa tutte le recensioni dell’utente
    - RemoveRecensione(index, ristorante): rimuove una recensione dalla lista (sia lato utente che ristorante)
    - ModificaRecensione(index, nuovoCommento, nuovoVoto, ristorante): aggiorna una recensione modificandola (prima la cancella, poi ne aggiunge una nuova)

3. Classe Ristoratore

La classe Ristoratore rappresenta un utente registrato con il ruolo di gestore di ristoranti. Estende la classe astratta Utente e fornisce funzionalità per gestire più ristoranti, leggere i dati da file, visualizzare recensioni e calcolare statistiche.

Attributi:

- ListaRistoranti: lista di oggetti Ristorante associati al ristoratore.
- Tutti gli attributi della classe Utente
- Ruolo: “Ristoratore”

Costruttori:

- Ristoratore(): inizializza un oggetto Ristoratore senza parametri.
- Ristoratore(String Nome, String Cognome, String Username, String Password, String Domicilio, int Giorno, int Mese, int Anno, boolean nuovo):
  accetta nome, cognome, username, password, domicilio, giorno, mese, anno di nascita, e un booleano per indicare se l'utente è nuovo.
- Ristoratore(String Nome, String Cognome, String Username, String Password, String Domicilio, String Data, boolean nuovo):
  accetta la data di nascita nel formato "gg-mm-aaaa" o "gg/mm/aaaa".

Metodi Principali:

- **Gestione del ristorante**
  - AggiungiRistorante: crea e aggiunge un nuovo ristorante alla lista del ristoratore con tutte le informazioni (nome, indirizzo, tipo cucina, coordinate geografiche, servizi, ecc.).
  - RimuoviRistorante: rimuove un oggetto Ristorante dalla lista.
  - getListaRistoranti: restituisce la lista dei ristoranti gestiti.
  - CaricaListaRistoranti: carica da file CSV i ristoranti associati all'utente specificato tramite username usando un oggetto GestoreRistoranti.
  - getRistorantiString: restituisce i nomi dei ristoranti separati da underscore (_) oppure "//" se la lista è vuota.
  - VisualizzaRistorante: restituisce una stringa contenente il nome del ristorante, la media stelle e il numero di recensioni.
- **Gestione delle recensioni:**
  - getRecensioniRistoranti: restituisce una stringa con tutte le recensioni dei ristoranti del ristoratore.

    - getRecensioniRistoranteSingolo(int index): restituisce una stringa con le recensioni di un ristorante specifico tramite l’indice nella lista.
    - getRecensioniRistoranteSingolo(String nomeRistorante): restituisce una lista di recensioni per il ristorante indicato tramite nome.
    - AggiungiRisposta: permette al ristoratore di aggiungere una risposta a una recensione, specificando l’indice e il nome del ristorante.
- **Metodi per le statistiche**
    - RecensioneMediaRistoranti: calcola e restituisce una stringa con la media delle valutazioni di ciascun ristorante e la media generale.
    - NumeroRecensioniRicevute: restituisce una stringa con il numero totale di recensioni per ogni ristorante gestito.

4. Classe UtenteNonRegistrato (Guest)

La classe UtenteNonRegistrato rappresenta un utente che non ha ancora effettuato la registrazione al sistema. Estende la classe Utente e fornisce funzionalità limitate come la visualizzazione delle informazioni di un ristorante e delle sue recensioni. Offre anche un metodo per registrarsi creando un nuovo oggetto Cliente.
Attributi: Non possiede attributi

Costruttore:

- Costruttore senza parametri, inizializza un oggetto UtenteNonRegistrato con valori predefiniti che identificano chiaramente un utente ospite:
  nome, cognome, username, password e domicilio impostati a valori vuoti o a "Guest"
- ruolo: "Utente non registrato"
- data di nascita: 1 gennaio 1900
- campo nuovo: impostato a false

Metodi:

- getDettagli(Ristorante ristorante): Ritorna una stringa con le informazioni dettagliate di un ristorante passato come parametro. Questo metodo chiama internamente il metodo visualizzaRistorante() della classe Ristorante.
- getRecensioni(Ristorante ristorante): Restituisce una lista di oggetti Recensione associati al ristorante indicato come parametro. Chiama internamente il metodo getRecensioni() dell'oggetto Ristorante.
- Registrare(...): Permette la registrazione di un nuovo cliente accettando tutti i dati anagrafici necessari:
  - nome
  - cognome
  - username
  - password
  - domicilio
  - giorno, mese e anno di nascita

5. Classe Ristorante

La classe Ristorante rappresenta un ristorante con tutte le sue informazioni, fornendo metodi per gestire recensioni, visualizzare dettagli e confrontare ristoranti.

Parametri:

- Nome: nome del ristorante.
- Nazione: nazione dove si trova il ristorante.
- Citta: città del ristorante.
- Indirizzo: indirizzo completo.
- TipoDiCucina: tipologia di cucina offerta.
- Servizi : servizi offerti dal ristorante.
- URLWeb: sito web del ristorante.
- Prezzo: indicazione generica del prezzo.
- FasciaDiPrezzo: fascia di prezzo in valori numerici.
- Stelle: valutazione in stelle (come stringa).
- Delivery: se offre servizio di consegna a domicilio.
- PrenotazioneOnline: se accetta prenotazioni online.
- Latitudine e Longitudine: coordinate geografiche.
- ContaRecensioni: contatore recensioni.
- MediaStelle: media dei voti delle recensioni.
- ListaRecensioni: lista di recensioni associate.

Costruttori:

- Costruttore vuoto
  Permette di creare un ristorante senza inizializzare attributi.
- Costruttore con parametri
  Inizializza tutti gli attributi passati come parametri, crea la lista recensioni vuota e calcola la media stelle.

6. Classe Gestore Ristoranti

La classe **GestoreRistoranti** rappresenta il componente responsabile della gestione dell'elenco di ristoranti e delle relative recensioni. Carica i dati da file all'avvio, consente l'accesso e la modifica della lista dei ristoranti e gestisce la scrittura persistente su file. Include anche funzionalità di ricerca avanzata tramite filtri.

Attributi:

- private ArrayList<Ristorante> listaRistoranti: lista dei ristoranti gestiti.
- private String FilePath: percorso del file CSV contenente i dati dei ristoranti.
- private String filePathRecensioni: percorso del file CSV contenente le recensioni.

Costruttore:

- GestoreRistoranti(): Costruttore senza parametri. Inizializza la lista dei ristoranti leggendo dal file FilePath e carica le recensioni corrispondenti da filePathRecensioni.

Metodi:

- getRistorante(String nome): Ristorante. Restituisce il ristorante che corrisponde esattamente al nome specificato. Se non viene trovato, restituisce null.
- getListaRistoranti(): ArrayList. Restituisce l'intera lista dei ristoranti attualmente caricati in memoria.
- AggiungiRistorante(Ristorante r): void. Aggiunge un nuovo ristorante alla lista.
- RimuoviRistorante(Ristorante r): void. Rimuove il ristorante specificato dalla lista.
- filtraPerNomeRistorante(String nome): ArrayList.. Restituisce un elenco di ristoranti filtrati in base al nome fornito. Cerca corrispondenze iniziali; se non trova risultati, cerca anche corrispondenze parziali.
- filtraPerCitta(String citta): ArrayList. Filtra i ristoranti in base alla città, prima localmente, poi applicando geocoding tramite libreria JOpenCageGeocoder per normalizzare il nome della città e ampliare i risultati.

- filtraPerTipoECitta(String tipo, String citta): ArrayList. Restituisce una lista di ristoranti che corrispondono sia al tipo di cucina specificato che alla città indicata.
- scriviSuFile(): void. Scrive su file sia i dati aggiornati dei ristoranti che tutte le recensioni associate. Utilizza i percorsi FilePath e filePathRecensioni.

Metodi privati:

- leggiDaFile(String FilePath): ArrayList. Metodo di supporto che legge i dati da un file CSV e restituisce una lista di oggetti Ristorante. Gestisce la corretta conversione dei campi, parsing di numeri, booleani e controllo del numero di colonne.

7. Classe Recensione

Classe che rappresenta una recensione per un ristorante. Contiene voto, commento, username dell'autore, nome del ristorante e una possibile risposta del ristoratore. Supporta recensioni con o senza risposta e consente di modificare i dati e confrontare due recensioni.

Attributi

- Commento: Testo della recensione dell'utente
- Username: Nome utente autore della recensione
- NomeRistorante: Nome del ristorante recensito
- Risposta: Risposta del ristoratore, vuota se assente
- Voto: voto intero espresso nella recensione (1-5)

Costruttori

- Recensione(int Voto, String Commento, String Username, String NomeRistorante)
  - Crea una recensione senza risposta.
  - Parametri: voto, commento, username autore, nome ristorante.
  - Risposta inizializzata a stringa vuota.
- Recensione(int Voto, String Commento, String Username, String NomeRistorante, String Risposta)
  - Crea una recensione completa con risposta.
  - Parametri: voto, commento, username autore, nome ristorante, risposta.

Metodi Pubblici

Getter

- String getNomeRistorante(). Restituisce il nome del ristorante recensito.
- String getCommento(). Restituisce il testo del commento.
- int getVoto(). Restituisce il voto assegnato.
- String getRisposta(). Restituisce la risposta del ristoratore (stringa vuota se assente).
- String getUsername(). Restituisce l'username dell'autore.

Setter / Modificatori

- void setCommento(String newCommento). Modifica il commento.
- void setVoto(int newVoto). Modifica il voto.
- void setRisposta(String newRisposta). Modifica o imposta la risposta.
- void EliminaRisposta(). Rimuove la risposta (imposta a stringa vuota).

Visualizzazione

- String visualizzaRecensione(). Restituisce una stringa leggibile della recensione.. Se non c'è risposta, stampa autore, voto e commento. Altrimenti aggiunge anche la risposta.
- @Override String toString(). Restituisce stringa completa con tutti i campi, risposta inclusa anche se vuota.

Confronto

- @ Override boolean equals(Object object). Confronta questa recensione con un altro oggetto. Restituisce true solo se: L'oggetto è una Recensione e voto, commento e risposta coincidono (stringhe con .equals). Ignora Username e NomeRistorante nel confronto. Restituisce false altrimenti.

8. Classe DataDiNascita

La classe DataDiNascita rappresenta una data valida di nascita, gestendo il controllo sulla correttezza del formato e sull'intervallo accettabile (tra il 1° gennaio 1900 e la data odierna). Fornisce metodi per ottenere giorno, mese, anno, età e per verificare se l'età è sufficiente per specifici requisiti (come 14 o 18 anni).

Attributi

- int Giorno, Mese, Anno: componenti della data.
- static final LocalDate MIN_DATE: data minima accettabile (01/01/1900).
- static final LocalDate MAX_DATE: data massima accettabile (la data odierna).

Costruttori

- DataDiNascita(int Giorno, int Mese, int Anno). Crea un oggetto DataDiNascita da valori numerici. Solleva IllegalArgumentException se la data è fuori intervallo o non valida (es. 31/02/2000).
- DataDiNascita(String data). Accetta una stringa nel formato "gg-mm-aaaa" o "gg/mm/aaaa". Solleva IllegalArgumentException in caso di formato errato o valori non validi.

Metodi Pubblici

- int getGiorno(), int getMese(), int getAnno(). Getter per le singole componenti della data.
- String getDataDiNascita(). Restituisce la data in formato "gg-mm-aaaa", con zeri iniziali per uniformità (es. 05-04-2002).
- int getEta(). Calcola e restituisce l'età in anni, tenendo conto del giorno e mese corrente per valutare se il compleanno è già stato superato nell'anno in corso.
- static boolean etaValida(int giorno, int mese, int anno). Verifica se la persona ha almeno **14 anni** rispetto alla data odierna. Ritorna false se la data è invalida.
- static boolean maggiorenne(int giorno, int mese, int anno). Verifica se la persona è **maggiorenne** (18 anni compiuti). Ritorna false in caso di data invalida.

Metodi Privati

- private boolean dataValida(int giorno, int mese, int anno). Verifica che la data esista (esclude giorni errati come il 30 febbraio) e che sia compresa tra MIN_DATE e MAX_DATE.

Eccezioni

Tutti i costruttori sollevano IllegalArgumentException in caso di formato errato della data, valori numerici invalidi, data fuori intervallo ammesso.

9. Classe Indirizzo

La classe Indirizzo si occupa dell'interazione con l'utente per ottenere un indirizzo preciso e valido a partire da un input testuale. Utilizza il servizio **JOpenCageGeocoder** per effettuare il geocoding dell'indirizzo e filtrare i risultati più rilevanti.

Costanti e Attributi

- API_KEY: stringa contenente la chiave API personale per il servizio JOpenCage.
- in: scanner pubblico statico usato per leggere l'input da tastiera. Utilizzato direttamente nei metodi statici per evitare creazioni multiple.

Costruttore

- Indirizzo(): costruttore vuoto di default. Non inizializza campi, dato che tutta la logica è gestita tramite metodi statici.

Metodo statico getSelezionaIndirizzo(String indirizzoInput)

Questo è il cuore della classe. Esegue le seguenti operazioni:

1. Creazione delle richieste:
    - Crea due richieste forward al geocoder:
        - una con l'indirizzo originale;
        - una "rilassata" (senza numeri) per catturare possibili alternative meno precise ma utili.
2. Recupero dei risultati:
    - Ottiene fino a 5 risultati per ciascuna richiesta, in lingua italiana e senza annotazioni geografiche extra.
3. Filtraggio:
    - Elimina:
        - risultati duplicati o contenuti uno nell'altro;
        - risultati troppo generici (es. solo città o regioni, senza numero civico).
4. Ordinamento:
    - I risultati validi vengono ordinati per lunghezza crescente: i più dettagliati appaiono per primi.

5. Interazione con l’utente:
    - Stampa una lista numerata dei risultati filtrati e chiede all’utente di selezionarne uno;
    - in alternativa, l’utente può scegliere l’opzione “0” per mantenere l’indirizzo originale inserito.
6. Restituzione:
    - Ritorna l’indirizzo selezionato (formattato dal geocoder), o quello originale se l’utente ha selezionato “0”.

10. Classe Input Annullato Exception

Questa classe definisce un'eccezione personalizzata che estende RuntimeException. Serve a segnalare in modo esplicito che l'utente ha annullato un input durante una procedura interattiva.

Costruttore

- InputAnnullatoException(). Costruttore senza parametri. Inizializza l'eccezione con il messaggio standard:

Dettagli tecnici

- Estende RuntimeException, quindi non richiede di essere dichiarata o gestita esplicitamente (unchecked exception).
- Usata tipicamente in metodi dove l'utente può scegliere di interrompere o uscire da un processo, ad esempio scrivendo una parola chiave come "annulla" o premendo ESC.

Vantaggi d'uso

- Migliora la leggibilità del codice segnalando chiaramente un caso d'uso specifico (annullamento volontario).
- Rende il flusso del programma più modulare: chi la cattura può gestire l'interruzione con messaggi mirati o rollback.