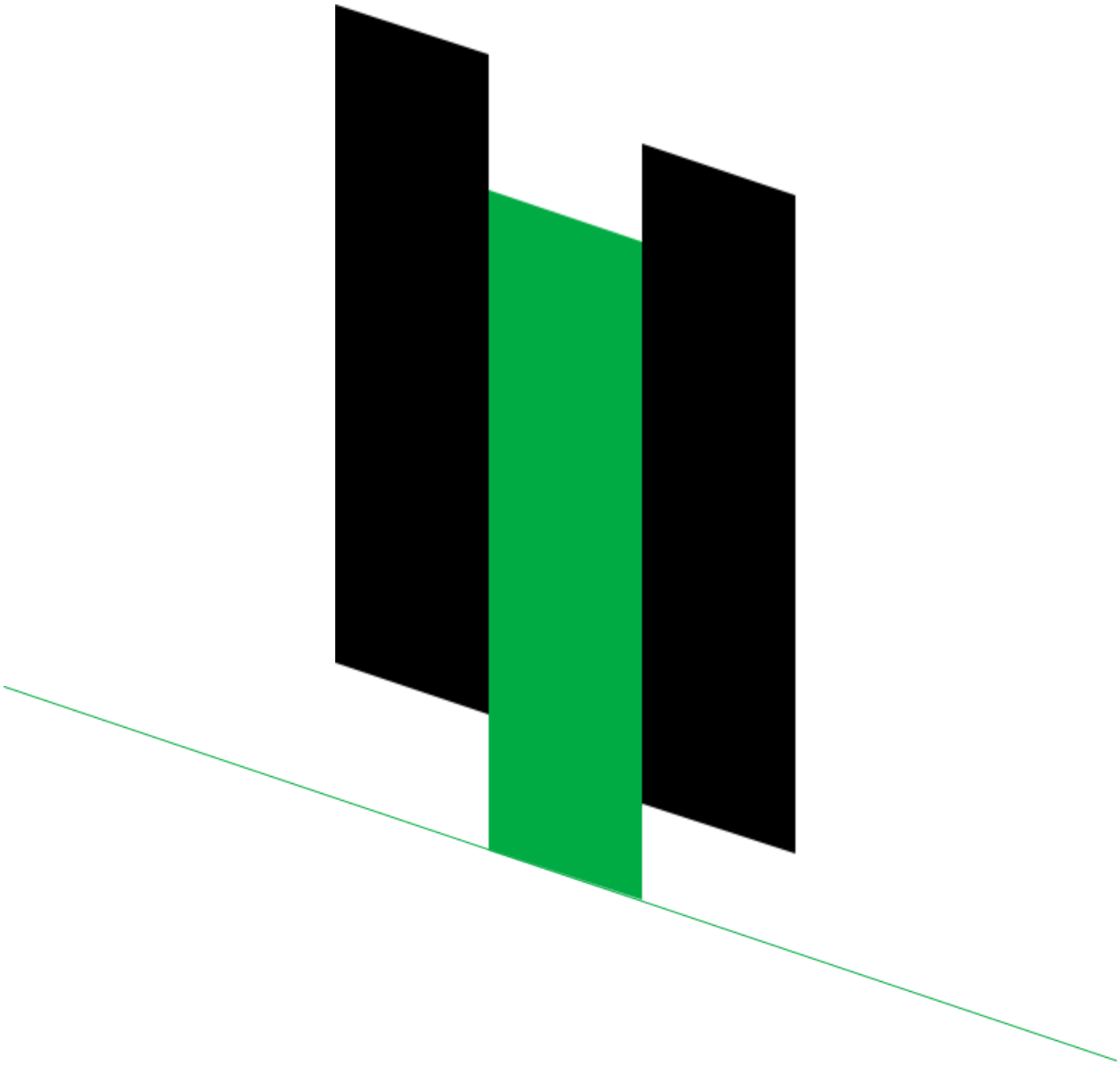

The Knife

Manuale tecnico

Progetto laboratorio A - università degli studi Insubria

Laurea triennale informatica

Sebastiano Svezia - Davide Bruno - Leonardo Bighetti - Francesco Vieri

INDICE

INTRODUZIONE

Il progetto *TheKnife* è un'applicazione console per la gestione e la ricerca di ristoranti, strutturata in modo da servire due diversi tipi di utenti: **clienti** e **ristoratori**. I clienti hanno la possibilità di cercare locali, gestire recensioni e salvare l'elenco dei ristoranti preferiti, mentre i ristoratori possono registrare i locali, visualizzare statistiche aggregate e rispondere alle recensioni. Il sistema si basa sulla persistenza dei dati tramite file di testo (`ristoranti.txt`, `recensioni.txt`, `utenti.txt`), e utilizza un set di classi strutturate sui seguenti pacchetti:

- `src.theknife` (interfaccia utente e logica di navigazione)
- `src.dao` (accesso ai dati e logica di business)
- `src.dto` (definizione dei modelli dati)
- `src.mapper` (mappatura dei dati tra file e oggetti)
- `src.sicurezzaPassword` (algoritmo di criptazione semplice)

STRUTTURA GENERALE DELL'APPLICAZIONE

L'architettura del sistema è modulare e si basa sulla separazione delle responsabilità:

- **Interfaccia utente:** gestita dalla classe `theknife` (package `src.theknife`), che fornisce il menu principale e coordina le interazioni utente.
- **Business Logic e Accesso ai Dati:** implementati nella classe DAO `GestioneTheKnife` (package `src.dao`), che esegue operazioni sui file di testo.
- **Modelli Dati (DTO):** definiti in `utente`, `ristorante` e `Recensione` (package `src.dto`).
- **Mapping:** il package `src.mapper` contiene metodi per convertire da stringhe a oggetti Java e viceversa.
- **Sicurezza:** implementata nel package `src.sicurezzaPassword` tramite metodi di criptazione semplici per la protezione delle password.

SCELTE ARCHITETTURALI

Le principali scelte architetturali includono:

- **Modularità:** separazione chiara tra interfaccia, logica di business e accesso ai dati.
- **DAO Pattern:** isolare l'accesso ai file di testo in un unico modulo, facilitando eventuali cambiamenti (come passare a un database relazionale).
- **Mapper Pattern:** utilizzo di funzioni di mappatura per abbassare il coupling tra la struttura interna degli oggetti e il formato di persistenza.
- **Robustezza:** gestione estesa degli errori tramite blocchi *try/catch* per garantire la stabilità dell'applicazione in presenza di file mancanti o dati malformati.

STRUTTURE DATI E FORMATI DI PERSISTENZA

Modelli Dati (DTO)

- `utente`**:** contiene proprietà quali nome, cognome, username, password (criptata), data di nascita (gestita tramite `Calendar`), domicilio, ruolo e una lista di ristoranti preferiti.
- `ristorante`**:** memorizza informazioni come nome, username del ristoratore, nazione, città, indirizzo, coordinate (latitudine e longitudine), prezzo e la disponibilità di servizi (delivery e prenotazione).
- `Recensione`**:** raccoglie il nome utente del recensore, il nome del ristorante, la valutazione (1-5), il testo della recensione e la risposta (eventuale).

Formato dei File e Gestione dei Path

- `ristoranti.txt`**:** I campi sono separati dal carattere `;`. Ogni riga contiene i dati di un ristorante (nome, ristoratore, nazione, città, indirizzo, coordinate, prezzo, disponibilità di servizi, tipo di cucina).
- `recensioni.txt`**:** I campi sono separati dalla virgola. Include username, dati del locale (composto da nome e luogo), valutazione, testo e risposta.

- `utenti.txt`**:** Utilizza la virgola per separare i campi (nome, cognome, username, password crittografata, data di nascita formattata – "yyyy-MM-dd", domicilio, ruolo e preferiti). I preferiti sono ulteriormente separati da un punto e poi da un punto e virgola per ogni dettaglio.

I path dinamici consentono al programma di trovare correttamente i file (es. `dati/utenti.txt`) sia durante lo sviluppo che in fase di esecuzione, senza dover specificare percorsi assoluti. Questo migliora portabilità e flessibilità tra ambienti (IDE, script `.bat`, JAR, ecc.).

Dopo la compilazione del progetto, l'applicazione può essere distribuita come file `.jar` (Java ARchive). Questo rende il programma eseguibile su qualsiasi sistema con Java installato, senza dover usare l'IDE.

Per semplificare l'avvio, si può utilizzare anche un file `.bat` su Windows, che lancia automaticamente il JAR.

SCELTE ALGORITMICHE E PATTERN APPLICATI

Algoritmi

- **Cifratura delle password:** L'algoritmo di criptazione adottato è basato su uno **shift ASCII** (metodo "Criptazione.critta()") che sposta ogni carattere di un numero fisso (CHIAVE) di posizioni. L'algoritmo inverso, "Criptazione.decritta()", ripristina il valore originale. Esempio di implementazione:

```
public static String critta(String testoChiaro) {
    String risultato = "";
    for (int i = 0; i < testoChiaro.length(); i++) {
        char c = testoChiaro.charAt(i);
        risultato += (char) (c + CHIAVE);
    }
    return risultato;
}
```

- **Parsing e validazione degli input:** Numerosi metodi (come `leggiNumero()`, `leggiBoolean()`, `parseDataNascita()`) sono stati implementati per validare l'input da console, garantendo che i dati inseriti soddisfino i requisiti di formato e siano coerenti. Ad esempio, il metodo di parsing della data:

```
private static Calendar parseDataNascita(String inputData) {
    if (inputData == null || inputData.trim().isEmpty()) {
        return new GregorianCalendar(0, 0, 1);
    } else {
        try {
            SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
            sdf.setLenient(false);
            Calendar cal = Calendar.getInstance();
            cal.setTime(sdf.parse(inputData));
            return cal;
        } catch (ParseException e) {
            System.out.println("Formato data non valido. Usa dd/MM/yyyy.
            return null;
        }
    }
}
```

-

Pattern Utilizzati

- **Pattern DAO (Data Access Object):** Il pattern DAO viene applicato nella classe `GestioneTheKnife`, separando la logica di accesso ai dati dalla logica di presentazione. Ciò semplifica eventuali cambiamenti nel meccanismo di persistenza (ad esempio, se si decidesse di passare da file di testo a un database).
- **Mapper Pattern:** Il package `src.mapper` implementa metodi per la conversione di stringhe in oggetti (e viceversa), ad esempio:

```java
public static ristorante mapObjRistorante(String line) {
    // Verifica e parsing dei campi
    String[] campi = line.split(";");
    if (campi.length < 11) return null;
    // Recupero dei dati e creazione dell'oggetto
    // ...
    return r;
}
```

- **Gestione Errori e Logging:** L'uso esteso di blocchi try/catch garantisce una robusta gestione degli errori durante operazioni I/O e parsing.

REQUISITI TECNICI

Questa sezione descrive l'ambiente tecnico e i requisiti necessari per lo sviluppo, l'esecuzione e la manutenzione dell'applicazione *TheKnife*.

Requisiti Hardware

- **Processore:** Qualsiasi processore compatibile con architettura x86/x64 (Intel, AMD) è sufficiente, poiché l'applicazione è leggera e non intensiva in termini computazionali.
- **Memoria RAM:** Minimum 4 GB; raccomandato almeno 8 GB per garantire un'esperienza fluida durante lo sviluppo e test.
- **Spazio di archiviazione:** Uno spazio inferiore a 500 MB è più che sufficiente, in quanto l'applicazione utilizza file di testo per la persistenza.

Requisiti Software

- **Sistema Operativo:** L'applicazione è indipendente dalla piattaforma ed è testata su Windows, macOS e Linux.
- **Java Runtime Environment (JRE):** È necessario avere installato almeno **Java SE 11** o una versione superiore, la quale garantisce il supporto delle funzionalità del linguaggio e la compatibilità con le API usate.
- **Ambiente di Sviluppo Integrato (IDE):** Consigliato l'uso di IntelliJ IDEA, Eclipse o Visual Studio Code per facilitare la gestione del progetto e la navigazione nel codice.
- **Librerie e Dipendenze:** Il progetto non utilizza librerie esterne complesse; si basa in modo esclusivo sulle API standard di Java.

- **Editor di Documenti:** Per la documentazione tecnica, si consiglia l'uso di **Google Documenti** con il font *Proxima Nova* per mantenere una formattazione moderna e leggibile.

Requisiti di Rete e Ambiente di Esecuzione

- **Connessione di rete:** Non essendo l'applicazione basata su servizi remoti o API, non è richiesta una connessione Internet costante per il suo funzionamento.
- **Ambiente Locale:** L'applicazione è destinata ad essere eseguita in locale (stand-alone), ideale per ambienti didattici o per piccoli prototipi.

LIMITI DELLA SOLUZIONE SVILUPPATA

Pur essendo strutturata in modo modulare e facilmente manutenibile, la soluzione presenta alcuni limiti:

- **Persistenza tramite file di testo:** Limita la scalabilità e la velocità delle operazioni in presenza di grandi quantità di dati.
- **Cifratura semplice:** L'algoritmo di shift ASCII per la protezione delle password non è adatto ad ambienti dove la sicurezza sia prioritaria.
- **Interfaccia utente testuale:** Limitata in termini di usabilità rispetto a una GUI (Graphical User Interface).
- **Gestione input:** Alcuni scenari limite potrebbero richiedere ulteriori controlli e validazioni per evitare crash o comportamenti inaspettati.

SITOGRAFIA / BIBLIOGRAFIA

- **Oracle Java Documentation:** https://docs.oracle.com/javase/
- **Wikipedia - DAO Pattern:** https://it.wikipedia.org/wiki/Data_Access_Object
- **Risorse online per Java:** StackOverflow, GitHub, forum e comunità Java.
- **Documentazione interna:** JavaDoc integrato nel codice sorgente.