THEKNIFE - APPLICAZIONE CONSOLE PER GESTIONE RISTORANTI

Manuale Tecnico

MEMBRI DEL GRUPPO

Nome e Cognome: **Kevin TCHIDJO FOTSO**
Numero Matricola: **755906**

Nome e Cognome: **Alessandro MASSANOVA**
Numero Matricola: **760228**

Università degli Studi dell'Insubria – **Laurea Triennale in Informatica**

Progetto Laboratorio A**: THE KNIFE**

Docente: **Prof. Loris Bozzato**

Anno Accademico: **2024-2025**

INTRODUZIONE

Questo manuale descrive l'architettura e le scelte progettuali di TheKnife. È destinato agli sviluppatori e amministratori di sistema che desiderano comprendere il funzionamento interno.

1. Architettura del Sistema

1.1 Panoramica Generale

TheKnife è un'applicazione Java desktop con architettura a tre livelli:

- **Presentation Layer**: Interfaccia console gestita dalla classe TheKnife
- **Business Logic Layer**: Logica applicativa nel GestoreDati
- **Data Access Layer**: Persistenza su file CSV

1.2 Pattern Architetturali Utilizzati

- **MVC Pattern**: Separazione tra presentazione (TheKnife), modello (classi dati) e controllo (GestoreDati)
- **Data Access Object (DAO)**: GestoreDati fornisce astrazione per l'accesso ai dati
- **Singleton Pattern**: Scanner globale per input utente

1.3 Struttura del Package

```
theknife/
├── TheKnife.java          # Entry point e UI
├── GestoreDati.java       # Business logic e persistenza
├── Ristorante.java        # Modello dati ristorante
├── Utente.java            # Modello dati utente
├── Recensione.java        # Modello dati recensione
├── Ruolo.java             # Enum per ruoli utente
└── AsciiTable.java        # Utility formattazione tabelle
```

2. Modello Dati

2.1 Strutture Dati Principali

Map<Integer, Ristorante> ristoranti

- **Chiave**: ID univoco ristorante (auto-incrementale)

- **Valore**: Oggetto Ristorante completo
- **Tipo**: LinkedHashMap per mantenere ordine di inserimento

Map<String, Utente> utenti

- **Chiave**: Username (univoco)
- **Valore**: Oggetto Utente con password hash
- **Tipo**: LinkedHashMap

Map<Integer, Recensione> recensioni

- **Chiave**: ID recensione (auto-incrementale)
- **Valore**: Oggetto Recensione
- **Tipo**: LinkedHashMap

Map<String, Map<Integer, String>> preferiti

- **Struttura nidificata**: username → (ristoranteId → nota)
- **Permette note personalizzate per ogni preferito**

2.2 Relazioni tra Entità

Utente (1) -----> (*) Recensione
Utente (1) -----> (*) Ristorante [se ruolo=RISTORATORE]
Utente (1) -----> (*) Preferito
Ristorante (1) -----> (*) Recensione
Recensione (1) -----> (0..1) Risposta [campo string]

2.3 Vincoli di Integrità

- **Username univoco** per utenti
- **ID auto-incrementali** per ristoranti e recensioni
- **Stelle 1-5** per recensioni
- **Una sola risposta** per recensione (campo stringa, non entità separata)
- **Ristoranti modificabili solo dal proprietario**
- **Recensioni modificabili solo dall'autore**

3. Gestione della Persistenza

3.1 Formato File CSV

Tutti i dati vengono persistiti in file CSV nella directory ../data/ relativa all'eseguibile:

utenti.csv

```
username,password_hash,nome,cognome,data_nascita,domicilio,ruolo
kevinf,0ffe1abd1a08215353c233d6e009613e95eec4253832a761af28ff37ac5a150c,kevin,fotso,1987-05-15,pavia,CLIENTE
marco_chef,edee29f882543b956620b26d0ee0e7e950399b1c4222f5de05e06425b4c995e9,marco,baldini,1990-07-20,Milano,RISTORATORE
M.Alessandro,0ffe1abd1a08215353c233d6e009613e95eec4253832a761af28ff37ac5a150c,Alessandro,Massanova,2023-05-06,Varese,CLIENTE
g.russo,edee29f882543b956620b26d0ee0e7e950399b1c4222f5de05e06425b4c995e9,ghislin,russo,1985-04-12,Parigi,RISTORATORE
b.loris,edee29f882543b956620b26d0ee0e7e950399b1c4222f5de05e06425b4c995e9,loris,Bozzato,1988-08-13,Genova,RISTORATORE
pcaruso,edee29f882543b956620b26d0ee0e7e950399b1c4222f5de05e06425b4c995e9,parson,caruso,1985-01-23,Berlino,RISTORATORE
mconti,edee29f882543b956620b26d0ee0e7e950399b1c4222f5de05e06425b4c995e9,moyo,conti,1990-05-03,Kyoto,RISTORATORE
```

ristoranti.csv

```
id,nome,nazione,citta,indirizzo,lat,lon,prezzo_medio,delivery,prenotazione,tipo_cucina,proprietario
1,Ristorante Il Duomo,Italia,Milano,Corso Vittorio Emanuele 3,454642.0,91900.0,35.0,false,true,Italiana,marco_chef
2,Bistro Lumi��re,Francia,Parigi,12 Rue de la Paix,488686.0,23320.0,34.0,true,false,Francese,g.russo
3,La Maison du Go3t,Francia,Lione,5 Rue Merci��re,457640.0,48357.0,32.0,true,true,Francese,g.russo
4,Osteria del Porto,Italia,Genova,Via al Mare 12,444048.0,89444.0,28.0,true,false,Italiana,b.loris
```

recensioni.csv

```
id,ristorante_id,username,stelle,testo,risposta
1,1,kevinf,4,ottimo cibo!,Grazie per la recensione!
2,4,M.Alessandro,4,Vista mare,consigliato al tramonto
3,3,m.giorgia,5,Specialit@ pesce,
4,2,m.giorgia,3,degustazione interessante,Grazie
5,10,m.giorgia,3,dolci da migliorare,
```

preferiti.csv

```
username,ristorante_id,note
M.Alessandro,4,
m.giorgia,3,
s.matteo,5,
s.matteo,9,
```

3.2 Parser CSV Personalizzato

Il sistema utilizza un parser CSV custom per gestire:

- **Campi con virgole**: Racchiusi tra virgolette

- **Virgolette nei dati**: Escape con doppie virgolette
- **Campi vuoti**: Gestiti correttamente
- **Righe vuote**: Ignorate

3.3 Gestione Errori I/O

- **IOException**: Catturate e riportate all'utente
- **File mancanti**: Creati automaticamente al primo salvataggio
- **Directory mancanti**: Create automaticamente
- **Salvataggio automatico**: Shutdown hook per persistenza garantita

4. Algoritmi di Ricerca e Ordinamento

4.1 Algoritmo di Ricerca Ristoranti

```
public List<Ristorante> cercaRistorante(parametri...) {
    return ristoranti.values().stream()
        .filter(r -> filtroGeografico(r))
        .filter(r -> filtroCucina(r))
        .filter(r -> filtroPrezzo(r))
        .filter(r -> filtroServizi(r))
        .filter(r -> filtroMedia(r))
        .collect(Collectors.toList());
}
```

Complessità: O(n) dove n = numero ristoranti totali

Filtri Implementati:

1. **Geografico**: Ricerca case-insensitive in città e nazione
2. **Cucina**: Match esatto case-insensitive
3. **Prezzo**: Range minimo/massimo inclusivo
4. **Servizi**: Boolean match per delivery/prenotazione
5. **Media recensioni**: Calcolata real-time

4.2 Calcolo Media Recensioni

```java
public double mediaRistorante(int ristoranteId) {
        int sum=0, n=0;
        for (Recensione r : recensioni.values()) if (r.ristoranteId==ristoranteId) { sum+=r.stelle; n++; }
        return n==0 ? 0.0 : (double)sum/n;
    }
```

Complessità: O(m) dove m = numero recensioni totali

Ottimizzazione possibile: Cache delle medie con invalidazione

4.3 Algoritmi di Ordinamento

- **Per prezzo**: Comparator.comparingInt(r->r.prezzoMedio)
- **Per nome**: Comparator.comparing(r->r.nome.toLowerCase())
- **Per media**: Comparator.comparingDouble(r->mediaRistorante(r.id))

Utilizza Collections.sort() con complessità O(n log n).

5. Sicurezza

5.1 Gestione Password

- **Hash SHA-256**: Implementazione con MessageDigest
- **Salt**: Non implementato (miglioramento futuro)
- **Storage**: Solo hash memorizzato, mai password in chiaro

```java
public static String sha256(String s) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            byte[] b = md.digest(s.getBytes("UTF-8"));
            StringBuilder sb = new StringBuilder();
            for (byte x : b) sb.append(String.format("%02x", x));
            return sb.toString();
        } catch (Exception e) { throw new RuntimeException(e); }
    }
```

5.2 Controllo Accessi

- **Autenticazione**: Username/password richiesti per operazioni sensibili
- **Autorizzazione**:

    - Clienti: Solo proprie recensioni e preferiti
    - Ristoratori: Solo propri ristoranti e loro recensioni
- **Validazione input**: Controlli su range valori (stelle 1-5, etc.)

5.3 Sanitizzazione Dati

- **CSV Injection**: Escape automatico di virgolette e caratteri speciali
- **Input validation**: Controlli su formati date, range numerici
- **Null safety**: Gestione valori nulli nei CSV

6. Gestione Errori e Eccezioni

6.1 Strategia di Error Handling

- **I/O Errors**: Try-catch con messaggi user-friendly
- **Parsing Errors**: Validazione dati con fallback sicuri
- **Business Logic Errors**: Valori di ritorno boolean/null per operazioni fallite
- **Runtime Errors**: Shutdown hook garantisce salvataggio dati

6.2 Validazione Input Utente

```
private static int leggiIntRange(String messaggio, int min, int max) {
    while (true) {
        int v = leggiInt(messaggio);
        if (v < min || v > max) System.out.println("Valore fuori range ("+min+"-"+max+").");
        else return v;
    }
}
```

Input robusto con loop fino a valore valido.

6.3 Gestione Stati Inconsistenti

- **ID referenze**: Controlli esistenza prima di operazioni
- **Relazioni**: Cascade delete non implementato (by design)
- **Transazioni**: Non supportate (singolo utente, file system)

7. Performance e Scalabilità

7.1 Complessità Operazioni

Operazione	Complessità	Note
Login utente	O(1)	HashMap lookup
Cerca ristorante	O(n)	Scan completo con filtri
Aggiungi recensione	O(1)	insert in HashMap
Media ristorante	O(m)	Scan recensioni
Salva dati	O(n+m+p)	Write tutti gli oggetti

7.2 Limitazioni Scalabilità

- **Memory-based**: Tutti i dati in RAM
- **Single-threaded**: Un utente alla volta
- **File locking**: Non implementato
- **Limite pratico**: ~10K ristoranti, ~100K recensioni

7.3 Ottimizzazioni Implementate

- **LinkedHashMap**: Ordine preservato con accesso O(1)
- **Stream API**: Lazy evaluation per filtri
- **Auto-incremento ID**: Evita scansioni per ID univoci
- **Shutdown hook**: Salvataggio garantito anche con interruzione

8. Estensibilità e Manutenzione

8.1 Punti di Estensione

- **Interfaccia utente**: Facile migrazione da console a GUI (Swing/JavaFX)
- **Persistenza**: Sostituibile con database relazionale (JDBC)
- **Autenticazione**: Integrabile con sistemi esterni (LDAP, OAuth)
- **Ricerca**: Estendibile con full-text search (Lucene)
- **Notifiche**: Aggiungibili email/SMS per nuove recensioni

8.2 Refactoring Possibili

```java
// Attuale: logica UI mista con business logic
private static void menuCliente(Utente u) {
    // UI + business logic insieme
}

// Miglioramento: separazione responsabilità
interface ClienteController {
    void gestisciPreferiti(String username);
    void gestisciRecensioni(String username);
}
```

8.3 Design Patterns Applicabili

- **Command Pattern**: Per azioni utente reversibili
- **Observer Pattern**: Per notifiche su nuove recensioni
- **Strategy Pattern**: Per algoritmi di ordinamento diversi
- **Factory Pattern**: Per creazione oggetti da CSV

9. Testing e Debug

9.1 Strategie di Testing

Unit Testing (non implementato, raccomandato):

```java
@Test
public void testCercaRistorante() {
    GestoreDati db = new GestoreDati(tempDir);
    // Popola dati test
    List<Ristorante> risultati = db.cercaRistorante("Milano", null, null, null, null, null, null);
    assertEquals(expectedCount, risultati.size());
}
```

Integration Testing:

- Test completo salvataggio/caricamento CSV
- Test integrità referenze tra entità
- Test operazioni CRUD complete

9.2 Debug e Monitoring

Logging (da aggiungere):

```
private static final Logger logger = LoggerFactory.getLogger(GestoreDati.class);

public void salva() {
    logger.info("Inizio salvataggio dati...");
    // operazioni
    logger.info("Salvati {} ristoranti, {} utenti", ristoranti.size(), utenti.size());
}
```

Debug CSV (implementato):

```
public String dumpPreferitiCsv() {
    // Mostra contenuto raw per debug problemi formato
}
```

9.3 Profiling e Metriche

Memory Usage:

- Ristorante: ~200 bytes/oggetto
- Recensione: ~100 bytes/oggetto
- Stima totale: 1M ristoranti = ~200MB RAM

I/O Performance:

- Lettura CSV: O(file_size)
- Scrittura CSV: O(num_records)
- Collo di bottiglia: parsing linee CSV

10. Deployment e Configurazione

10.1 Build Process

Compilazione
javac -d bin -cp src src/theknife/*.java

Creazione JAR
jar cfe bin/TheKnife.jar theknife.TheKnife -C bin .

```
# Struttura finale
project/
├── bin/TheKnife.jar
├── data/            # File CSV
├── README.txt
└── src/theknife/  # classe Java
├── lib          #librerie esterne
├── README.txt
└── doc/
```

10.2 Configurazione Runtime

System Properties utilizzate:

- user.dir: Directory base per percorso dati
- file.encoding: Encoding file CSV (UTF-8 raccomandato)

Percorsi File:

```
Path base = Path.of(System.getProperty("user.dir"));
Path data = base.resolve("../data").normalize();
```

Assume esecuzione da directory bin/ con dati in data/ al livello superiore.

10.3 Requisiti di Deployment

- **Java Runtime**: JRE 8+
- **Permessi filesystem**: Lettura/scrittura directory dati
- **Memoria**: Minimo 128MB heap (più per dataset grandi)
- **Spazio disco**: Proporzionale ai dati (1MB per ~1000 ristoranti)

11. Migrazione e Versioning

11.1 Compatibilità CSV

Schema versioning (non implementato, raccomandato):

```
# Intestazione con versione
#version=1.0
username,password_hash,nome,cognome,data_nascita,domicilio,ruolo
```

Migrazione dati:

```
public class DataMigrator {
    public void migrateFromV1ToV2(Path dataDir) {
        // Logica conversione formato
    }
}
```

11.2 Backward Compatibility

Attualmente **non garantita** tra versioni. Raccomandazioni:

- Backup dati prima di aggiornamenti
- Script di migrazione per modifiche schema
- Validazione versione file all'avvio

12. Gestione Chiusura Applicazione

12.1 Shutdown Hook Implementation

L'applicazione implementa un meccanismo di salvataggio automatico tramite shutdown hook per garantire la persistenza dei dati:

```
    Runtime.getRuntime().addShutdownHook(new Thread(() -> {
    try {
        db.salva();
    } catch (Exception e) {
        System.err.println("Errore salvataggio: " + e.getMessage());
    }
}));
```

12.2 Scenari di Chiusura

Chiusura Normale

- **Trigger**: Selezione opzione "0) Esci" dal menu principale
- **Comportamento**: Return dal metodo menuIniziale(), esecuzione naturale dello shutdown hook
- **Salvataggio**: Automatico e garantito

Chiusura Forzata (Ctrl+C / SIGTERM)

- **Trigger**: Interruzione da terminale o kill del processo
- **Comportamento**: JVM esegue shutdown hook prima della terminazione
- **Salvataggio**: Automatico tramite hook

Chiusura Anomala (SIGKILL / OutOfMemory)

- **Trigger**: Kill forzato del processo o crash JVM
- **Comportamento**: Nessun shutdown hook eseguito
- **Rischio**: Perdita dati dell'ultima sessione

12.3 Gestione Errori in Chiusura

```
// Implementazione robusta dello shutdown hook
Runtime.getRuntime().addShutdownHook(new Thread(() -> {
    try {
        System.err.println("Salvataggio dati in corso...");
        db.salva();
        System.err.println("Dati salvati con successo.");
    } catch (Exception e) {
        System.err.println("ERRORE CRITICO: Impossibile salvare i dati!");
        System.err.println("Dettagli: " + e.getMessage());
        e.printStackTrace();
    }
}));
```

12.4 Raccomandazioni per Robustezza

- **Backup periodici**: Implementare salvataggio incrementale durante l'esecuzione
- **Transazioni**: Salvare dopo ogni operazione critica
- **Recovery**: Meccanismo di ripristino da backup in caso di corruzione
- **Validazione**: Controllo integrità dati al caricamento

13. Limitazioni Note e Miglioramenti Futuri

12.1 Limitazioni Architetturali

1. **Single-user**: Non supporta utenti concorrenti
2. **Memory-based**: Tutti i dati in RAM
3. **No transactions**: Rischio inconsistenza su errori I/O
4. **CSV limitations**: Problemi con caratteri speciali
5. **No backup**: Dati persi se corrotti

12.2 Miglioramenti Performance

- **Database relazionale**: Persistenza scalabile
- **Indici**: Per ricerche geografiche e full-text
- **Caching**: Media recensioni pre-calcolate
- **Paginazione**: Per risultati grandi
- **Connection pooling**: Se multi-user

12.3 Miglioramenti Funzionali

- **Ricerca geografica**: Raggio in km da coordinate
- **Upload immagini**: Per ristoranti
- **Sistema rating**: Più articolato (servizio, cibo, ambiente)
- **Filtri avanzati**: Prezzo per persona, orari apertura
- **Social features**: Follow altri utenti, condivisione liste

12.4 Miglioramenti Sicurezza

- **Salt per password**: Prevenire rainbow table attacks
- **Input sanitization**: Prevenire injection attacks
- **Session management**: Token-based authentication
- **Rate limiting**: Prevenire spam recensioni
- **Audit logging**: Tracciamento operazioni sensibili

13. API Reference

13.1 GestoreDati - Metodi Pubblici Principali

```
// Caricamento/Salvataggio
public void carica() throws Exception
public void salva() throws Exception

// Autenticazione
public Utente registrazione(String username, String password, String nome,
                            String cognome, String dataN, String domicilio, Ruolo ruolo)
public Utente login(String username, String password)

// Ricerca
public List<Ristorante> cercaRistorante(String paeseOCitta, String tipoCucina,
                                        Double prezzoMin, Double prezzoMax,
                                        Boolean delivery, Boolean prenotazione,
                                        Double mediaMin)

// Visualizzazione
public String visualizzaRistorante(Ristorante r)
public String visualizzaRecensioni(int ristoranteId)

// Operazioni Cliente
public boolean aggiungiPreferito(String username, int ristoranteId)
public boolean rimuoviPreferito(String username, int ristoranteId)
public Recensione aggiungiRecensione(String username, int ristoranteId, int stelle, String testo)
public boolean modificaRecensione(String username, int recensioneId, int nuoveStelle, String nuovoTesto)

// Operazioni Ristoratore
public Ristorante aggiungiRistorante(String proprietario, String nome, String nazione,
                                     String citta, String indirizzo, double lat, double lon,
                                     double prezzoMedio, boolean delivery, boolean prenotazione, String tipoCucina)
public boolean rispostaRecensioni(String proprietario, int recensioneId, String risposta)
```

13.2 Codici di Errore e Return Values

- **null**: Operazione fallita (login, registrazione username duplicato)
- **false**: Operazione non permessa (modifica recensione non propria)
- **true**: Operazione completata con successo
- **Lista vuota**: Nessun risultato trovato (ricerca)

Questa architettura fornisce una base solida per un sistema di gestione ristoranti con possibilità di evoluzione verso soluzioni più scalabili e feature-rich.

14. Bibliografia e Sitografia

14.1 Documentazione Tecnica di Riferimento

- **Oracle Java SE Documentation**: https://docs.oracle.com/en/java/javase/21/ - Documentazione ufficiale Java per API standard utilizzate (Collections, I/O, Security)
- **Java Language Specification (JLS)**: https://docs.oracle.com/javase/specs/ - Specifica formale del linguaggio Java

14.2 Standard e Specifiche

- **RFC 4180 - CSV Format**: https://tools.ietf.org/html/rfc4180 - Specifica formale formato file CSV
- **FIPS PUB 180-4**: Federal standard per algoritmi hash SHA-256
- **ISO/IEC 8859-1 (Latin-1)** e **UTF-8**: Standard di encoding caratteri per file di testo
- **IEEE 754**: Standard per rappresentazione numeri floating-point (tipo double)

14.3 Architettura e Design Patterns

- **Design Patterns (Gang of Four)**: Patterns architetturali MVC, DAO utilizzati nel progetto
- **Martin Fowler - Patterns of Enterprise Application Architecture**: Principi architetturali per applicazioni enterprise

14.4 Algoritmi e Strutture Dati

- **Introduction to Algorithms (CLRS)**: Riferimento per algoritmi di ordinamento e ricerca implementati
- **Java Collections Framework**: https://docs.oracle.com/javase/8/docs/technotes/guides/collections/ - Documentazione HashMap, LinkedHashMap utilizzate
- **Stream API Guide**: https://docs.oracle.com/en/java/javase/21/docs/api/index.html - API per operazioni funzionali su collezioni

14.5 Sicurezza Informatica

- **OWASP Top 10**: https://owasp.org/www-project-top-ten/ - Vulnerabilità comuni applicazioni
- **Secure Coding Guidelines :** https://www.oracle.com/java/technologies/javase/seccodeguide.html
- **Security Developer's Guide** : https://docs.oracle.com/en/java/javase/21/security/index.html/

14.6 Testing e Quality Assurance

- **JUnit Documentation**: https://junit.org/junit5/docs/current/user-guide/ - Framework testing utilizzabile per unit test
- **Software Testing Techniques**: Metodologie testing per validazione funzionalità

14.7 Persistenza Dati

- **File I/O Best Practices**: Java documentation per gestione file system
- **Database Design Principles**: Principi normalizzazione dati (per future evoluzioni)
- **NoSQL vs SQL**: Comparative analysis per scelte persistenza alternative