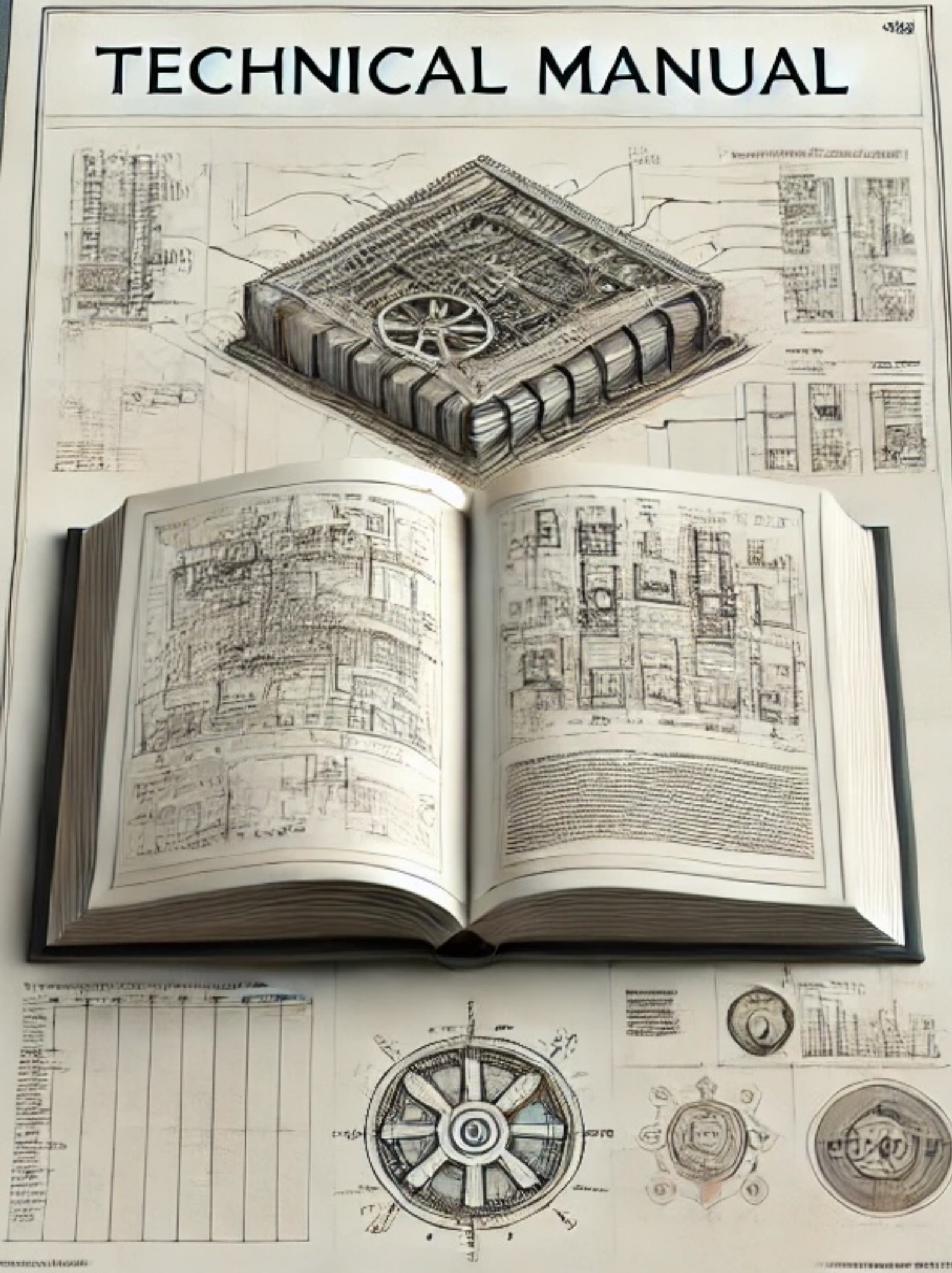
TECHNICAL MANUAL



Progetto a cura di: Gervasini Alessio, Orsini Francesco Pio, Borin Luca

Indice:

❖ Introduzione

Il progetto “Book Recommender” è un’applicazione progettata per gestire e raccomandare libri agli utenti. Il sistema permette la registrazione degli utenti, la creazione di librerie personali, la valutazione dei libri e la raccomandazione di nuovi titoli basati sulle valutazioni. Questo manuale tecnico fornisce una panoramica dettagliata della struttura del codice, delle librerie utilizzate, delle strutture dati impiegate, delle scelte tecniche e della complessità del codice.

❖Struttura del Codice (suddivisione classi)

Il progetto è suddiviso in diverse classi principali, ognuna con responsabilità specifiche:

1. Main.java

- Contiene il punto di ingresso dell'applicazione e gestisce il menu principale e le operazioni di login e registrazione degli utenti.

```java
public static void main(String[] args) {
    // Punto di ingresso del programma.
    // Chiama il metodo menu() per iniziare l'esecuzione.
}

public static void autoaggiornamento() {
    // Imposta un timer per controllare periodicamente se i file di valutazioni o
consigli esistono.
    // Se esistono, cancella il primo timer e avvia un secondo timer per aggiornare
i dati a intervalli regolari.
}

public static void menu() {
    // Mostra il menu principale e gestisce le scelte dell'utente.
    // Permette all'utente di cercare libri, registrarsi, fare login o uscire dal
programma.
}

public static void menuUtenteRegistrato(String userid) {
    // Mostra il menu per gli utenti registrati e gestisce le loro scelte.
    // Permette di creare una libreria, aggiungere libri, valutare libri, consi-
gliare libri o fare logout.
}

public static void ricercalibronologin() {
    // Mostra il menu di ricerca per gli utenti non registrati.
    // Permette di cercare libri per titolo, autore o autore e anno.
}

private static List<String[]> leggiFileCsv(String filePath) {
    // Legge un file CSV e restituisce una lista di array di stringhe contenenti i
dati.
}

public static boolean libroEsisteInDataCsv(String titolo) {
    // Controlla se un libro con il titolo specificato esiste nel file CSV di dati.
}

public static boolean isNumeric(String str) {
    // Controlla se una stringa è numerica.
}
```

- **FILE_PATH** e **LIBRERIE_FILE_PATH** sono costanti che rappresentano i percorsi dei file CSV utilizzati per memorizzare i dati dei libri e delle librerie.

- Il metodo main gestisce il menu principale e il menu utente registrato tramite cicli while e switch per gestire le scelte dell'utente.

2. RegistrazioneUtente.java

- Gestisce la registrazione e il login degli utenti.

```
public class RegistrazioneUtente {
    public static void registrazione() {
        // Logica per la registrazione degli utenti
    }

    public static String login() {
        // Logica per il login degli utenti
    }

    private static boolean isUserIDExists(String userID) {
        // Verifica se l'userID esiste già nel file CSV
    }

    // Altri metodi di supporto per la validazione e il salvataggio dei dati
}
```

- La registrazione include la validazione dell'userid, codice fiscale, e-mail e password.

- Il metodo isUserIDExists verifica se l'userID è già presente nel file UtentiRegistrati.csv.

- Il metodo login gestisce l'autenticazione degli utenti.

3. ConsiglioLibro.java

- Gestisce le raccomandazioni dei libri basate sulle valutazioni degli utenti.

```
public class ConsiglioLibro {
    private String titoloLibro;
    private List<String> libriConsigliati;
    private Utente utente;

    public String getTitoloLibro() {
        //Restituisce il titolo del libro consigliato
    }

    public List<String> getLibriConsigliati() {
        // Restituisce la lista dei libri consigliati
    }

    public Utente getUtente() {
        //Restituisce l'utente che ha fornito il consiglio
    }

    public static void writeToFile(String userId, String selectedBook,
      List<String> recommendedBooks) throws IOException {
        //Metodo statico per scrivere i consigli su un file csv
    }

}
```

- Fornisce accesso alle informazioni memorizzate nei campi privati dell'oggetto, come il titolo del libro consigliato, la lista dei libri consigliati e l'utente associato.

- Gestisce la scrittura delle informazioni relative al consiglio di un libro in un file CSV specifico, includendo l'ID dell'utente, il libro selezionato e fino a tre libri consigliati.

4. Liberia.java

- Gestisce la creazione e la gestione delle librerie personali degli utenti.

```
public class Libreria {
    private String nomeLibreria;
    private List<String> libri;

    public Libreria(String nomeLibreria) {
        // Inizializzazione della libreria
    }

    public boolean aggiungiLibro(String libro) {
        // Aggiunge un libro alla libreria se non già presente
    }

    public static void registraLibreria(String utente, Libreria libreria) {
        // Registra la libreria nel file CSV
    }

    public static String visualizzaLibrerieConLibri(String userid) {
        // Visualizza le librerie di un utente specifico
    }

    public static String getLibreriaByIndex(String userid, int index) {
        // Recupera il nome della libreria per indice
    }

    public static String getLibroByIndex(String userid, String nomeLibreria, int
index) {
        // Recupera il titolo del libro per indice
    }
}
```

- Utilizza un ArrayList per memorizzare i titoli dei libri in una libreria.

- I metodi visualizzaLibrerieConLibri, getLibreriaByIndex e getLibroByIndex aiutano a recuperare e visualizzare i dati delle librerie e dei libri.

- I dati delle librerie sono memorizzati in Librerie.dati.csv.

5. ValutazioneLibro.java

- Gestisce l'inserimento e l'aggiornamento delle valutazioni dei libri.

```
public class ValutazioneLibro {
    private static final String VALUTAZIONI_FILE_PATH = "ValutazioniLibri.csv";

    public static void inserisciValutazioneLibro(String titoloLibro, String userid)
{
        // Logica per inserire la valutazione di un libro
    }

    private static int inputValutazione(Scanner scanner) {
        // Metodo di supporto per l'inserimento delle valutazioni
    }

    public static void calcolaMedie() {
        // Calcola le medie delle valutazioni per ogni criterio
    }

    public static void scriviValutazioneCsv(String userid, String titoloLibro, int
stile, int contenuto, int gradevolezza, int originalita, int edizione, double
mediaValutazioni, String valutazione) {
        // Scrive i dati delle valutazioni nel file CSV
    }
}
```

- La classe gestisce l'inserimento e la memorizzazione delle valutazioni dei libri in un file CSV (ValutazioniLibri.csv). Fornisce metodi per inserire nuove valutazioni (inserisciValutazioneLibro) e calcolare le medie delle valutazioni per criterio (calcolaMedie).

- Include un metodo privato (inputValutazione) per garantire l'inserimento corretto delle valutazioni numeriche tramite un oggetto Scanner. Facilita l'interazione con l'utente durante l'inserimento dei dati delle valutazioni.

- Utilizza BufferedWriter e FileWriter per scrivere i dati delle valutazioni nel file CSV, assicurando la persistenza a lungo termine delle informazioni dei libri valutati.

6. Utente.java

- Definisce la struttura e i metodi relativi agli utenti.

```
public class Utente {
    private String userid;
    private String password;
    private String nome;
    private String cognome;
    private String codiceFiscale;
    private String email;

    private static Map<String, Utente> utentiRegistrati = new HashMap<>();

    public Utente(String userid, String password, String nome, String cognome,
String codiceFiscale, String email) {
       //Costruttore per inizializzare un nuovo oggetto Utente
    }
     public static void registraUtente(Utente utente) {
        // Metodo statico per registrare un nuovo utente.
        // Aggiunge l'utente alla Map utentiRegistrati.
    }
      public static Utente getUtente(String userid) {
        // Metodo statico per ottenere un utente dato il suo userid
    }
     public boolean autentica(String password) {
        // Metodo per autenticare l'utente controllando la password
    }
     // Metodi Getter per accedere alle informazioni dell'utente
     // Userid, Nome, Cognome, CodiceFiscale, E-mail
}
```

- Memorizza i dati personali di un utente (userid, password, nome, cognome, codice Fiscale, e-mail) come variabili di istanza.

- Utilizza una mappa statica (utentiRegistrati) per registrare nuovi utenti e recuperare utenti esistenti tramite il loro userid.

- Fornisce un metodo per autenticare l'utente confrontando la password fornita con quella memorizzata.

❖ Librerie Usate

Il progetto utilizza le seguenti librerie Java standard:

- java.io. *: Per operazioni di input/output, come la lettura e la scrittura di file.
- java. util. *: Per l'utilizzo di strutture dati come Map, List e Scanner.

❖ Strutture Dati utilizzate

Le strutture dati principali utilizzate nel progetto includono:

- ❖ ArrayList
- ❖ Map
- ❖ List
- ❖ HashSet
- ❖ HashMap
- ❖ LinkedHashMap
- ❖ Array

Di seguito sono riportati i vantaggi e gli svantaggi di ciascuna struttura dati, insieme alla motivazione del loro utilizzo specifico nel progetto.

○ ArrayList

• Descrizione della struttura

• ArrayList è una classe in Java che implementa l'interfaccia List e utilizza un array di dimensioni variabili per memorizzare gli elementi. Consente di memorizzare una collezione ordinata di elementi duplicati e offre metodi per accedere, inserire, rimuovere e manipolare gli elementi dell'array.

• Vantaggi

- **Accesso diretto agli elementi:** Gli elementi in un ArrayList possono essere acceduti direttamente tramite un indice, il che rende l'accesso molto veloce (complessità O(1)).

- **Operazioni di inserimento/rimozione efficaci:** Supporta l'inserimento e la rimozione degli elementi con un'efficienza elevata, specialmente quando viene aggiunto o rimosso un elemento alla fine dell'array (complessità O(1) ammortizzata).

- **Dimensione dinamica:** ArrayList si espande automaticamente quando viene aggiunto un numero maggiore di elementi rispetto alla capacità attuale.

• Svantaggi

- **Dimensionamento dinamico:** L'espansione dinamica richiede che l'array venga ridimensionato quando è pieno, il che può comportare un overhead di memoria e di tempo.

- **Costo elevato per operazioni di inserimento/rimozione in posizioni intermedie:** Se viene inserito o rimosso un elemento in posizioni intermedie dell'array, tutti gli elementi successivi devono essere spostati, il che può essere costoso in termini di tempo (complessità O(n)).

• Motivazione dell'utilizzo

- ArrayList è stato scelto per memorizzare la lista dei libri in Libreria per la sua capacità di gestire una collezione ordinata di libri con accesso rapido agli elementi tramite indice e per la sua flessibilità nell'aggiungere e rimuovere libri quando necessario.

○ Map

• Descrizione della struttura

- Map è un'interfaccia in Java che rappresenta una struttura dati di tipo chiave-valore, dove ogni elemento è una coppia di chiave e valore. In particolare, nel codice fornito, viene utilizzata LinkedHashMap, che è una classe che implementa l'interfaccia Map e mantiene l'ordine di inserimento delle chiavi.

• Vantaggi

- **Ricerca efficiente**: Le operazioni di ricerca per chiave sono molto efficienti (generalmente O(1) per le implementazioni hash come HashMap).

- **Struttura flessibile**: Permette di gestire dati strutturati in coppie chiave-valore, ideale per rappresentare associazioni logiche tra dati.

- ## Svantaggi

  - **Occupazione di memoria**: Alcune implementazioni, come HashMap, possono consumare più memoria rispetto ad altre strutture dati più semplici.

  - **Costo delle collisioni**: Se non gestite correttamente, le collisioni hash possono influenzare le prestazioni.

- ## Motivazione dell'utilizzo

  - Nel metodo visualizzaLibrerieConLibri per esempio, Map<String, List<String>> librerieUtente viene utilizzata per associare ogni nome di libreria ad una lista di libri associati a quell'utente. Questo permette di organizzare e rappresentare in modo efficace le librerie di un determinato utente, fornendo una struttura chiara e organizzata per la visualizzazione dei dati.

○ List

• Descrizione della struttura

- List è un'interfaccia che estende Collection e definisce un insieme ordinato di elementi che possono contenere duplicati. In Java, ci sono diverse implementazioni di List, come ArrayList, LinkedList, Vector, ecc., ognuna con caratteristiche e performance diverse.

• Vantaggi

- **Accesso per indice**: Gli elementi possono essere acceduti tramite un indice intero, il che rende l'accesso agli elementi molto veloce (O(1) per ArrayList).
- **Dimensione dinamica**: Le implementazioni di List in Java sono dimensionabili dinamicamente, il che significa che possono crescere o ridursi in base alle operazioni di aggiunta o rimozione degli elementi.
- **Metodi utili**: Fornisce molti metodi utili per manipolare gli elementi, come add, remove, get, indexOf, size, ecc.

• Svantaggi

- **Costo delle operazioni di inserimento/rimozione**: Alcune implementazioni (ArrayList in particolare) possono essere costose se è necessario inserire o rimuovere elementi in posizioni diverse dalla fine della lista.

- **Ricerca lineare**: Alcune operazioni come “contains” richiedono una ricerca lineare se non è noto l'indice dell'elemento, il che può essere meno efficiente in grandi collezioni di dati.

• Motivazione dell’utilizzo

- Nel codice della classe ConsiglioLibro per esempio, la struttura dati List viene utilizzata per gestire la lista dei libri consigliati associati a un libro selezionato da un utente. Questo è utile perché permette di mantenere una lista ordinata e dinamica di libri consigliati, che può essere facilmente modificata aggiungendo o rimuovendo libri in base alle raccomandazioni.

○ HashSet

• Vantaggi

- **Velocità di accesso**: Le operazioni di inserimento, rimozione e verifica della presenza di un elemento sono molto veloci (tempo medio costante, O(1)).

- **Unicità degli elementi**: Garantisce che ogni elemento aggiunto all'insieme sia unico.

- **Implementazione basata su hash**: Utilizza funzioni hash per organizzare gli elementi, ottimizzando l'accesso e la ricerca.

- **Metodi utili**: Fornisce metodi per iterare sugli elementi, verificare l'appartenenza di un elemento, unirsi ad altri set, ecc.

- ## Svantaggi

  - **Non ordinato**: Gli elementi all'interno di un HashSet non sono mantenuti in ordine specifico, il che potrebbe non essere ideale in certi scenari.

  - **Overhead di memoria**: Utilizza una tabella hash, quindi può consumare più memoria rispetto ad altre implementazioni di set, specialmente per insiemi di grandi dimensioni.

  - **Iterazione inefficace**: Se è necessario iterare attraverso gli elementi in un ordine specifico, HashSet non garantisce un ordinamento.

- ## Motivazione dell'utilizzo

  - Nel codice della classe Libricsv, per esempio, HashSet è utilizzato per memorizzare i consigli associati a ciascun libro. Questo è vantaggioso perché consente di gestire un insieme unico di consigli per ogni titolo di libro senza duplicati.

○ HashMap

• Descrizione della struttura

- HashMap in Java è una classe che implementa l'interfaccia Map, utilizzando una tabella hash per memorizzare coppie chiave-valore. È utilizzato per implementare mappe di dati, dove ogni chiave è unica e mappa a un singolo valore.

• Vantaggi

- **Velocità di accesso**: Le operazioni di inserimento, rimozione e ricerca (get, put, remove) sono molto veloci (tempo medio costante, O(1)) in condizioni ideali.

- **Struttura chiave-valore**: È ideale per la memorizzazione di dati strutturati come coppie chiave-valore, rendendo facile associare informazioni.

- **Adatto per grandi quantità di dati**: Gestisce efficacemente grandi quantità di dati senza compromettere le prestazioni.

- **Metodi utili**: Fornisce metodi per iterare sugli elementi, controllare la presenza di una chiave, ottenere le chiavi o i valori, ecc.

Svantaggi

- **Non ordinato**: Gli elementi all'interno di una HashMap non sono mantenuti in un ordine specifico, il che può essere un problema se è richiesto un certo ordine.

- **Overhead di memoria**: Usa una tabella hash internamente, quindi può consumare più memoria rispetto ad altre strutture dati meno complesse come gli array.

- **Collisioni**: Sebbene rara, una collisione hash può verificarsi, influenzando le prestazioni in casi specifici.

Motivazione dell'utilizzo

- Nel codice della classe Libricsv, per esempio, HashMap è utilizzato principalmente per due scopi:

    1. **Valutazioni dei libri**: Memorizza le valutazioni dei libri, utilizzando il titolo del libro come chiave e una lista di valori interi come valore.

    2. **Consigli per i libri**: Memorizza i consigli associati a ciascun libro, utilizzando il titolo del libro come chiave e un insieme di stringhe come valore.

○ LinkedHashMap

• Descrizione della struttura

- LinkedHashMap è una classe in Java che estende HashMap e mantiene l'ordine di inserimento dei suoi elementi. Oltre a fornire tutte le funzionalità di HashMap, mantiene una lista doppiamente collegata che collega tutti gli elementi in base all'ordine di inserimento.

• Vantaggi

- **Mantiene l'ordine di inserimento**: Conserva l'ordine in cui gli elementi sono stati inseriti. Questo è utile quando è necessario iterare sugli elementi in base all'ordine di inserimento.

- **Velocità di accesso**: Offre prestazioni simili a HashMap per le operazioni di inserimento, ricerca e rimozione degli elementi (tempo medio costante, O(1)).

- **Iterazione prevedibile**: Le iterazioni su un LinkedHashMap avvengono nell'ordine in cui gli elementi sono stati inseriti.

- **Metodi utili**: Fornisce metodi per accedere all'ultimo elemento inserito (getLastEntry()), rimuovere il primo elemento inserito (removeEldestEntry()), e altro ancora.

- ## Svantaggi

  - **Utilizzo di memoria**: Usa più memoria rispetto a una HashMap a causa della struttura aggiuntiva per mantenere l'ordine di inserimento.

  - **Complessità maggiore**: Le operazioni di inserimento, rimozione e riassegnazione possono richiedere più tempo rispetto a una HashMap standard a causa della gestione dell'ordine.

- ## Motivazione dell'utilizzo

  - Nel codice per il metodo visualizzaLibrerieConLibri, per esempio, LinkedHashMap è utilizzato per memorizzare le librerie di un utente insieme ai libri associati a ciascuna libreria. Mantenere l'ordine di inserimento delle librerie e dei loro libri è cruciale per garantire che vengano visualizzati nell'ordine corretto all'utente.

○ Array

• Descrizione della struttura

- Utilizzato per memorizzare le valutazioni dei libri.

• Vantaggi

- **Accesso Rapido**: Gli array offrono un accesso costante agli elementi tramite indice, con complessità O(1).

- **Efficienza di Memoria**: Gli array di tipi primitivi sono molto efficienti in termini di memoria.

• Svantaggi

- **Dimensione Fissa**: La dimensione degli array è fissa dopo la loro creazione, il che li rende meno flessibili rispetto alle liste.

- **Nessun Supporto per Operazioni Complesse**: Gli array non supportano operazioni complesse come l'aggiunta e la rimozione dinamica degli elementi.

• Motivazione dell'utilizzo

- Gli array sono stati utilizzati per memorizzare le valutazioni dei libri (double[] valutazioni) perché il numero di caratteristiche valutate è fisso (5), rendendo l'array una scelta efficiente sia in termini di accesso che di memoria.

❖ Scelte Tecniche (CSV vs TXT)

• Introduzione

Nel progetto “Book Recommender” è stata scelta l’estensione CSV (Comma-Separated Values) per la memorizzazione dei dati dei libri e delle valutazioni. Di seguito viene fornito un confronto tra i formati CSV e TXT, motivando perché l’uso di CSV è stato più ottimale in questo contesto.

○ CSV (Comma-Separated Values)

• Vantaggi

❖ **Struttura Tabellare**: I file CSV sono progettati per rappresentare dati tabellari. Ogni riga rappresenta un record e ogni colonna è separata da una virgola, facilitando l’organizzazione e la manipolazione dei dati.

❖ **Compatibilità**: I file CSV sono ampiamente supportati da molte applicazioni e strumenti di analisi dei dati come Excel, Google Sheets e database.

❖ **Semplicità di Parsing**: La struttura del CSV rende il parsing dei dati semplice e diretto. Molti linguaggi di programmazione, inclusi Java, Python, e C#, offrono librerie integrate per lavorare con i file CSV.

❖ **Efficienza**: I file CSV sono generalmente piccoli in termini di dimensioni, poiché non includono formattazioni aggiuntive, rendendo il trasferimento e la lettura dei file rapida.

- **Svantaggi**

  - **Assenza di Struttura Gerarchica**: I file CSV non possono rappresentare dati gerarchici o nidificati facilmente.

  - **Errori di Formattazione**: L'uso delle virgole come delimitatori può causare problemi se i dati stessi contengono virgole, richiedendo l'uso di escape characters o virgolette per delimitare i campi.

TXT (Plain Text)

- **Vantaggi**

  - **Flessibilità**: I file TXT possono contenere qualsiasi tipo di dato testuale, rendendoli estremamente flessibili.
  - **Semplicità**: Sono semplici da creare e modificare con qualsiasi editor di testo.

  - **Formato Libero**: Non c'è una struttura imposta, permettendo di organizzare i dati in qualsiasi modo necessario.

- **Svantaggi**

  - **Mancanza di Struttura**: La mancanza di una struttura definita rende difficile l'interpretazione automatica dei dati. Ogni programma che legge un file TXT deve conoscere il formato specifico dei dati.

- **Difficoltà di Parsing**: Estrarre informazioni strutturate dai file TXT può essere complesso, specialmente se i dati non seguono un formato rigoroso.

- **Incompatibilità**: I file TXT non sono facilmente importabili in strumenti di analisi dei dati senza una pre-elaborazione significativa.

• Motivazione per l'Utilizzo di CSV

Nel contesto del progetto "Book Recommender", l'uso del formato CSV è stato scelto per i seguenti motivi:

- **Struttura dei Dati**: I dati dei libri e delle valutazioni sono naturalmente tabellari, con ogni record (libro) contenente campi fissi come titolo, autore, anno, valutazioni, ecc. La struttura tabellare del CSV è perfetta per rappresentare questo tipo di dati.

- **Facilità di Manipolazione**: La presenza di librerie integrate in Java per la manipolazione dei file CSV semplifica enormemente la lettura, scrittura e aggiornamento dei dati.

- **Compatibilità**: I file CSV possono essere facilmente importati ed esportati da strumenti di analisi dei dati come Excel, il che è utile per la visualizzazione e l'analisi dei dati senza la necessità di un'elaborazione preliminare.

- **Efficienza**: I file CSV sono efficienti in termini di dimensioni e velocità di lettura/scrittura, il che è importante per mantenere le performance dell'applicazione.

In sintesi, l'uso del formato CSV offre una soluzione strutturata, compatibile ed efficiente per la memorizzazione e manipolazione dei dati nel progetto "Book Recommender", rendendolo una scelta ottimale rispetto al formato TXT.

❖Scelte Tecniche (utilizzo del Bucket Sort)

• Introduzione

Nel progetto “Book Recommender” è stato scelto di utilizzare il Bucket Sort per ordinare alfabeticamente la raccolta di Libri in ogni Libreria Utente in modo da garantire una maggiore efficienza in termini di ricerca. Di seguito viene fornita un’analisi che motiva il perché l’uso del Bucket Sort è stato più ottimale in questo contesto.

• Vantaggi:

1. **Efficienza con distribuzioni uniformi**: Il Bucket Sort può essere molto efficiente quando i dati sono distribuiti uniformemente su un intervallo noto, con una complessità temporale media che può essere vicina a O(n).
2. **Adatto per parallelizzazione**: Poiché i bucket possono essere ordinati in parallelo, l'algoritmo si presta bene all'ottimizzazione su sistemi multi-core. Poiché ogni core lavora in parallelo, il tempo complessivo per ordinare l'intero elenco può essere ridotto notevolmente rispetto a un algoritmo che deve ordinare tutti gli elementi in sequenza (uno alla volta).
3. **Ordinamento lineare**: In scenari ottimali, il Bucket Sort può raggiungere una complessità temporale lineare, O(n), rendendolo più veloce rispetto a molti algoritmi di ordinamento tradizionali come il Quick Sort (O(n log n)).
4. **Ordina mantenendo l'ordine relativo**: Bucket Sort è un algoritmo stabile, il che significa che gli elementi con valori identici rimangono nell'ordine relativo in cui erano prima dell'ordinamento.

• Svantaggi:

1. **Dipendenza dalla distribuzione dei dati**: Se i dati non sono distribuiti uniformemente, il Bucket Sort può perdere efficienza e ridursi a una complessità O(n^2).
2. **Necessità di memoria aggiuntiva**: L'algoritmo richiede spazio aggiuntivo per i bucket, il che può essere un problema quando lo spazio è limitato.

3. **Non adatto a tutti i tipi di dati**: È più efficace quando l'intervallo dei dati è noto e limitato. Con dati complessi o con range vasti, potrebbe non essere la scelta migliore.

- ## Motivazione dell'utilizzo:

Utilizzare il Bucket Sort può essere vantaggioso rispetto ad altri algoritmi di ordinamento quando si ha a che fare con un grande set di dati che è distribuito uniformemente su un intervallo noto. In queste condizioni, il Bucket Sort può ordinare i dati in tempo lineare O(n), superando l'efficienza di algoritmi come il Quick Sort o il Merge Sort, che hanno una complessità media di O(n log n). Questo rende il Bucket Sort ideale per il contesto utilizzato.

❖ Esempio di Complessità del Codice

Il metodo aggiornaLibriCsv() nella classe ValutazioneLibro.java è un esempio di complessità del codice. Questo metodo legge il file Libri.csv, cerca il libro specificato, calcola e aggiorna le medie delle valutazioni per stile, contenuto, gradevolezza, originalità ed edizione, e riscrive il file con le nuove informazioni.

La complessità in tempo del metodo è principalmente O(n), dove n è il numero di valutazioni lette e scritte. Mentre la complessità in spazio è O(n), "n" rappresenta il numero di righe presenti nel file Libri.csv.

❖ Grafica Programma

Il programma attualmente non dispone di una interfaccia grafica (GUI) e tutte le operazioni vengono effettuate tramite interfaccia a riga di comando (CLI). Per migliorare l'esperienza utente, potrebbe essere utile sviluppare una GUI utilizzando librerie come JavaFX o Swing.

❖ Conclusione

Il progetto "Book Recommender" è un sistema completo per la gestione e la raccomandazione dei libri, implementato con un'architettura modulare e utilizzando file CSV per la memorizzazione dei dati. Per un futuro sviluppo, sarebbe utile considerare l'integrazione di un database e l'implementazione di una GUI per migliorare l'esperienza utente.