Manuale tecnico

Autori: Zhang Ying Huang, matricola 746483; Alessandro Di Lorenzo, matricola 733052
Data: 03 settembre 2022
Versione documento: 1.0

- Indice

- Struttura dell'applicazione

- Package emotionalsongs

- EmotionalSongs (Main)
- DataHandler
- Feedback
- Emotions
- Song
- Author

- Package utente

- AbstractUser
- Address
- AddressQualifier
- LoggedUser
- NotLoggedUser
- Playlist

- EmotionalSongs

Questa classe contiene il metodo main, l'entry point del programma.
Contiene le varie HashMap elencate di seguito, dove vengono caricati i dati dei rispettivi file:

- reg_users: contiene i dati degli utenti registrati, presi dal file UtentiRegistrati.csv;
- system_songs: continene i dati della repository di canzoni di sistema, presi dal file Canzoni.csv;
- uid_plnameANDsongs: contiene i dati delle playlist dei singoli utenti, presi dal file Playlist.csv;
- song_emotions: contiene i dati delle valutazioni per le varie emozioni delle singole canzoni, presi dal file Emozioni.csv;

Contiene, inoltre, vari metodi di appoggio.

Ad esempio, il menù, è diviso in 3 parti:

- il metodo showBaseMenu stampa su console le opzioni disponibili per un utente qualsiasi;

- il metodo showNotLoggedMenu stampa su console le opzioni disponibili solamente per gli utenti non autenticati;
- il metodo showLoggedMenu stampa su console le opzioni disponibili solamente per gli utenti autenticati.

Poi, ci sono una serie di metodi scritti al fine di snellire il main e renderlo più comprensibile, che verranno riportati e approfonditi nella sezione delle Scelte architetturali.

- DataHandler

Questa classe contiene le funzioni dedicate esclusivamente ad interagire con i file e fa da ponte tra questi ultimi e le hashmaps.
La classe contiene metodi che si occupano di:

- inizializzare i file, qualora non siano presenti;
- salvare su file eventuali modifiche apportate ai dati;
- caricare i dati dei vari file nelle rispettive hashmaps;

Inoltre, contiene anche i path di ogni file con cui interagisce il programma.
Ulteriori spiegazioni tecniche e dettagli verrano trattati nella sezione apposita.

- Feedback

Questa classe è la rappresentazione di una valutazione legata ad un'emozione provata per una singola canzone. Un feedback è identificato univocamente dalla coppia id della canzone ed emozione, perciò possono esserci nove occorrenze di feedback per una canzone, dove cambia l'emozione.
Un oggetto feedback possiede, inoltre, un punteggio cumulativo e un numero totale di recensioni, con i quali poi si calcola la media del punteggio dato ad una specifica emozione provata per una determinata canzone. Inoltre, è possibile associare un commento personale.
Ulteriori spiegazioni tecniche e dettagli verrano trattati nella sezione apposita.

- Emotions

Questa classe è la rappresentazione di un'emozione che può essere associata ad ogni singola canzone. Nel caso di questo programma, è una classe enumerativa, contenente 9 costanti enumerative e le loro relative descrizioni:

- AMAZEMENT;
- SOLEMNITY;
- TENDERNESS;
- NOSTALGIA;
- CALMNESS;
- POWER;
- JOY;
- TENSION;
- SADNESS.

Ulteriori spiegazioni tecniche e dettagli verrano trattati nella sezione apposita.

- Song

Questa classe è la rappresentazione di un brano, a cui viene associato un autore, un id univoco e un anno di pubblicazione. Ogni canzone viene poi ricostruita al livello del programma come un oggetto di tipo Song. Rappresenta una delle componenti principali su cui agisce l'intera applicazione.
Ulteriori spiegazioni tecniche e dettagli verrano trattati nella sezione apposita.

- Author

Questa classe è la rappresentazione di un autore che può essere associato ad un brano. Ogni oggetto autore è identificato da un nome ed un cognome.
Un oggetto autore non può avere solo un nome, ad esempio, un nome d'arte, ma deve per forza avere uno e un solo nome ed uno e un solo cognome, in quanto dotato di due campi: firstName e lastName.
Ulteriori spiegazioni tecniche e dettagli verrano trattati nella sezione apposita.

- AbstractUser

Questa classe è la rappresentazione di un utente generico e viene estesa dalle classi LoggedUser e NotLoggedUser. Contiene le funzioni che sono comuni ad entrambi i tipi di utente.
Ulteriori spiegazioni tecniche e dettagli verrano trattati nella sezione apposita.

- NotLoggedUser

Questa classe è la rappresentazione di un utente non autenticato, il quale può effettuare le seguenti operazioni:

- cercare una canzone, per nome o per autore e anno;
- registrarsi;
- autenticarsi.

Ulteriori spiegazioni tecniche e dettagli verrano trattati nella sezione apposita.

- LoggedUser

Questa classe è la rappresentazione di un utente autenticato, il quale può effettuare le seguenti operazioni:

- cercare una canzone, per nome o per autore e anno;
- creare una playlist;
- inserire un brano in una data playlist;
- inserire una valutazione per un'emozione provata ascoltando un determinato brano;
- uscire dal profilo e tornare al menù per gli utenti non autenticati.

Ulteriori spiegazioni tecniche e dettagli verrano trattati nella sezione apposita.

- Address

Questa classe è la rappresentazione di un indirizzo fisico, costituito da:

- qualificatore (via, viale o piazza);
- nome della via;
- numero civico;
- città;
- provincia.

Costituisce un campo appartenente all'utente autenticato, da fornire durante la registrazione.
Ulteriori spiegazioni tecniche e dettagli verrano trattati nella sezione apposita.

- AddressQualifier

Questa classe è la rappresentazione di un qualificatore di un indirizzo fisico; in questo caso, è una classe enumerativa con 3 costanti:

- VIA;
- VIALE;
- PIAZZA;

Ulteriori spiegazioni tecniche e dettagli verrano trattati nella sezione apposita.

- Playlist

Questa classe è la rappresentazione di una playlist appartenente ad un utente. Un oggetto di tipo playlist ha un nome ed una lista di brani ed è possibile aggiungere brani ad una playlist già esistente.
Ulteriori spiegazioni tecniche e dettagli verrano trattati nella sezione apposita.

- Scelte architetturali

L'applicazione è suddivisa in due package con lo scopo di favorirne la manutenzione e di agevolarne lo sviluppo in un'ottica di ambiente concorrente. Il flusso dei dati da un package all'altro, avviene attraverso la classe EmotionalSongs.

- Package emotionalsongs: al suo interno le classi si occupano interamente della gestione e manipolazione dei dati. Le classi Song, Author, Emotions e Feedback sono in stretta relazione tra loro: I loro metodi, sono principalmente implementati con lo scopo di fornire un prospetto riassuntivo per un determinato brano.
  L'idea, che sta dietro la struttura di queste classi, è che il concetto di "prospetto riassuntivo per un singolo brano" è realizzato da un insieme finito di oggetti di tipo feedback, di cardinalità compresa tra [0,9]. La dimensione variabile di questo insieme di feedback è dovuta al fatto che, nella classe enumerativa Emotions, esistono 9 costanti (tag emozionali). Un singolo oggetto feedback a sua volta, è composto da:
    - Una costante Emotions e l'identificativo del brano (questa coppia di valori è la chiave che consente l'indicizzazione del feedback nella HashMap 'song_emotions' presente nella classe EmotionalSongs.).

    - La media delle valutazioni (compresa tra 1 e 5, realizzata dalla divisione [totalNumberOfScores / score], dove: - this.totalNumberOfScores è il numero totale di valutazioni - this.score è la somma di tutti i valori di tutte le valutazioni.

    - I commenti lasciati dagli utenti (recensire è facoltativo). Questi a loro volta sono contenuti nel campo notes, HashMap<userID, commento>. La ragione di usare una HashMap sta nel fatto che controllare il valore delle chiavi impedisce allo stesso utente di valutare più volte lo stesso brano per quell'emozione. Infine la classe Song, attraverso il suo metodo getFeedback(), restituisce la LinkedList contenente tutti i feedback a lei associati. La lista, una volta passata alla classe EmotionalSongs, verrà visualizzata a schermo.

- Package utente: in questo package, le classi definiscono i metodi volti a realizzare le funzionalità di un utente. Le istanze di questa classe comunicano con EmotionalSongs, completando quindi il canale di comunicazione.
  L'organizzazione interna segue questa ripartizione logica:
    - la gerarchia di classi AbstractUser, LoggedUser e NotLoggedUser realizza le funzionalità di ricerca dei brani, di login e logout. I motivi

che hanno portato alla creazione di questa gerachia sono fondamentalmente due:

- Crea una netta separazione tra le funzionalità messe a disposizione tra le diverse tipologie di utente.

- Facilita l'evoluzione dell'applicazione (in termini di concorrenza, distribuzione, introduzione di database, aggiunta di nuove funzionalità ecc..). Questo secondo motivo ha tuttavia portato la classe NotLoggedUser ad avere, oltre il metodo login(String seralizedUser), il costruttore che inizializza il campo guestID, che non viene direttamente utilizzato. Facendo però un trade off di queste considerazioni, si è capito che i benefici di questa architettura ne superavano i costi (in termini di poco codice).

Scendendo nei dettagli della gerarchia, la classe astratta AbstractUser viene utilizzata per passare, in esecuzione, da un'istanza di NotLoggedUser ad una di LoggedUser e viceversa. Inoltre, definisce in overloading il metodo per la ricerca dei brani che restituisce una LinkedList contenente oggetti di tipo Song (risultati dalla ricerca). Infatti, findSong(String title) ricerca per titolo un brano presente nella HashMap system_songs e findSong(String author, int year) invece, effettua tale ricerca tramite autore e anno. Entrambi restituiscono un oggetto LinkedList<Song> contenente i risultati della ricerca.

- la classe Playlist, realizza le funzionalità per la gestione dei brani contenuti all'interno delle sue istanze. La classe LoggedUser ha come attributo una LinkedList di questi oggetti e ne modifica i valori e contenuti usufruendo dei metodi che la classe mette a disposizione.

- le classi Address e AddressQualifier, vengono utilizzate per svolgere l'attività di registrazione di un nuovo utente all'applicazione.

- Scelte algoritmiche

Nell'applicazione vengono utilizzate principalmente queste due strutture dati: HashMap e LinkedList. Le HashMap sono pensate per contenere tutta l'informazione dei file (che vengono letti all'avvio), in modo tale che l'accesso a tali informazioni sia in tempo O(1) invece che O(n), caso che si verificherebbe se ogni volta venisse letto il

file in esecuzione. Tutte le HashMap sono quindi presenti nel package emotionalsongs. Le LinkedList al contrario, vengono utilizzate principalmente nel package utente. Ciò è dovuto al fatto che un utente possiede un insieme di playlist di cardinalità variabile, compresa tra [0,N]. Tra le scelte prese in fase di analisi dei requisiti, si è deciso di non mettere alcun limite al numero di playlist che un'istanza di LoggedUser può creare. A loro volta, le istanze di Playlist contengono un numero variabile di oggetti Song. Questo valore è compreso tra [1,K], dove K è il numero di brani presenti nella repository di sistema. Il valore minimo di cardinalità è dovuto al fatto che si è deciso di non permettere la creazione di playlist vuote. Per quanto riguarda invece i tempi di esecuzione complessivi dei metodi di sistema, tra i quali ricerca di brani e visualizzazione dei relativi feedback, la complessità in termini di tempo è 3*O(N), (quindi O(N)). Questa è la stessa complessità di ogni metodo presente in EmotionalSongs che esegue la stampa su console dei risultati. Ciò è inevitabile, perchè stampare a schermo il risultato di una ricerca richiede un iterazione dei suoi elementi. Altre operazioni: come la login, la verifica di esistenza di una playlist per un certo utente e l'aggiornamento dei valori delle hashmap, avviene in O(1).

- Formato dei file e la loro gestione

La classe DataHandler si occupa interamente della gestione di tutti i file. All'avvio, carica il contenuto di ogni file nelle rispettive HashMap che sono istanziate nella classe EmotionalSongs. Nello specifico, il metodo INIT_PROGRAM_FILES() crea, qualora non esistessero, la cartella "data" e al suo interno i file: Emozioni.csv, Playlist.csv, UtentiRegistrati.csv, Canzoni.csv. I file, sono tutti del formato .csv perchè l'informazione che contengono non è di per sé leggibile, cioè non è una frase di senso compiuto, ma una serie di dati: Nei metodi loadRegUsers(), loadSongs(), loadEmotion(), loadPlsts() di DataHandler, per ogni riga del file, il metodo .split("regex") viene invocato per separarne i diversi componenti, ciascuno dei quali rappresenta parte di un'informazione o di un oggetto che verrà istanziato. Una volta che l'informazione viene caricata nelle corrispondenti strutture dati del programma (ed elaborata/utilizzata), al terminare della sua esecuzione, viene poi ri-serializzata e riscritta sui rispettivi file.

- JavaDoc

Il codice sorgente è commentato tutto in formato javadoc e le pagine html generate dal comando javadoc sono presenti nella cartella compressa javadoc.zip.

- Limiti della soluzione sviluppata

- L'applicazione non è integrata con una GUI al momento.;
- Non è presente la concorrenza, perciò può esserci solo un utente alla volta ad interagire col programma e non è possibile, per esempio, aggiornare concorrentemente un dato;
- Non è un programma distribuito, perciò non possono esserci più istanze dello stesso programma che interagiscono tra di loro in una rete;
- Non è presente un'interazione con un database; i dati sono strettamente conservati sul file in maniera persistente;
- La presenza di più HashMap a livello del programma potrebbe rendere il programma pesante al crescere della mole di dati da trattare;
- Conseguentemente al punto appena precedente, il salvataggio di tutti i dati dalle HashMap ai file, durante l'esecuzione, potrebbe essere molto lento, in quanto in ogni "case" dello switch presente nel main, c'è un operazione di salvataggio del contenuto delle hashmap nei file;
- Anche l'interazione con i file, al crescere della mole di dati, potrebbe rallentare il programma, soprattutto in funzione di quanto scritto nel punto sopra;
- Possibile ridondanza dei dati data dalla mancanza di un database.

Tutti i limiti elencati verrano risolti/limitati e le rispettive funzioni implementate nella versione successiva dell'applicazione.

- Stilografia/Biblografia

Università dell'Insubria (sede Como)
Corso di Informatica

Profili github:
Zhang Ying Huang
Alessandro Di Lorenzo