UNIVERSITÀ DEGLI STUDI DI PISA

Corso di Programmazione II

Corso A

A.A 2019/2020

Documentazione

Primo Progetto-sessione autunnale

Autore:

Lorenzo Angeli

539036

Introduzione

In questo documento verrà presentato lo scopo del progetto della sessione autunnale del corso di Programmazione II per l'A.A 2019/2020; la sua struttura generale e le varie scelte di implementazione.

1 Progetto

In questa sezione verrà presentata l'implementazione effettiva del progetto e il motivo di alcune scelte di implementazione. Si noti che è stata completamente mantenuta l'interfaccia DataBoard<E extends Data> e i metodi utilizzati in aggiunta sono stati inseriti solamente nelle classi ad eccezione del metodo Display().

1.1 Strutture e scelte implementative

Il sistema è stato realizzato decidendo di dividere l'implementazione in tre blocchi principali: il primo rappresenta la classe di tipo Data, il secondo la singola categoria all'interno della collezione dati e il terzo rappresenta invece la collezione dati stessa.

La prima entità che prendiamo in considerazione è la classe Data<E> che implementa l'interfaccia Comparable<Data<E>>. La descrizione di un dato avviene attraverso:

- Il **Tipo di dato**, caratterizzato da una variabile generica E.
- Il **Numero di like** del dato, caratterizzato da una variabile di tipo int.
- La **Collezione dei contatti** (amici) che hanno messo like al dato, caratterizzata da una HashMap<String, Integer>.

La Collezione dei contatti (amici) serve a mantenere l'insieme delle persone che hanno aggiunto like al dato in modo tale da poter controllare se il contatto (amico) che ha aggiunto like non lo abbia già fatto in precedenza. Avrei potuto utilizzare un'ArrayList al posto della HashMap, ma la seconda struttura risulta essere più efficiente della prima. Questa mappa memorizza coppie (chiave, valore). Utilizzo una stringa String come chiave e come valore un intero Integer anche se questo non ci servirà.

La seconda entità che prendiamo in considerazione è **Categoria**<E extends Data<?>> che implementa l'interfaccia Category<E extends Data<?>>. La descrizione di ogni singola categoria avviene attraverso la tripla:

- Il **Nome della Categoria**, caratterizzato da una variabile di tipo String.
- La **Collezione di dati**, caratterizzata da un'ArrayList di dati di tipo E.
- La **Collezione dei contatti** (amici), caratterizzata da un'ArrayList di dati di tipo String.

Il primo elemento è fondamentale in quanto identifica la categoria all'interno della collezione, di conseguenza non può assumere valore nullo.
Per quanto riguarda la collezione dei dati è stata scelta una struttura di tipo ArrayList perché con strutture dati come HashTable in caso di pochi elementi si poteva generare spazio inutilizzato.

Sempre riguardo agli elementi che compongono la collezione di dati è importante sottolineare come quest'ultima può contenere dei duplicati. Il terzo elemento rappresenta l'insieme degli amici che hanno accesso in sola lettura ai dati della collezione, questo tipo di collezione non accetta duplicati. All'interno della classe Categoria è stato implementato l'override del metodo equals per poter confrontare due oggetti di tipo Categoria.

La terza entità che prendiamo in considerazione è **Board**<T, E extends Data<T>> che implementa DataBoard<E extends Data<?>>. La descrizione di questa classe avviene attraverso:

- La **Password**, caratterizzata da una variabile di tipo String.
- La **Collezione delle categorie**, caratterizzata da una collezione di dati di tipo Categoria.

Nella collezione in questione non ci possono essere categorie ripetute.

Come richiesto dalla consegna del progetto la seguente classe è stata implementata utilizzando due strutture dati differenti. La prima implementazione è stata fatta mediante un'**ArrayList** < *Categoria<E>* > al quale si può accedere tramite la Password disponibile al proprietario della Bacheca. La seconda implementazione è stata fatta mediante una struttura che implementa tabelle con le tabelle Hash, ovvero un **HashMap** < String, *Categoria<E>* > dove la chiave primaria rappresenta il nome della Categoria. In questo modo è banale distinguere le varie Categorie e si previene la possibilità di avere duplicati Tra le due implementazioni è preferibile l'utilizzo della seconda per una migliore efficienza.

1.2 Eccezioni

L'intero progetto si basa su una programmazione di tipo difensiva, che prevede l'utilizzo di una serie di eccezioni per ogni metodo che è stato implementato. Il controllo è ovviamente fatto in modo che l'invariante di rappresentazione venga rispettato. Nel caso non lo sia, viene sollevata l'eccezione adeguata. In particolar modo si fanno uso di sette eccezioni:

- **CategoriaNonPresenteException**, che viene lanciata se vengono riscontrati problemi relativi alla mancata presenza di una Categoria.
- **CategoriaPresenteException**, che viene lanciata se vengono riscontrati problemi relativi alla presenza di una Categoria.
- **DatoNonPresenteException**, che viene lanciata se vengono riscontrati problemi relativi alla mancata presenza di un dato.
- **FriendNonPresenteException**, che viene lanciata se vengono riscontrati problemi relativi alla mancata presenza di un amico.
- **FriendPresenteException**, che viene lanciata se vengono riscontrati problemi relativi alla presenza di un amico.

- **PasswordErrataException**, che viene lanciata se vengono riscontrati problemi relativi alla componente *password* del proprietario della Bacheca.
- **LikePresenteException**, che viene lanciata se il contatto (amico) ha già messo like in precedenza a uno specifico dato.

1.3 Invariante di rappresentazione

Vengono presentati in questa parte del documento i tre invarianti di rappresentazione: uno per Data, uno per Category e l'altro per Board.

2.3.1 Data

Overview: tipo di dato modificabile che rappresenta un oggetto che contiene un dato generico al quale è possibile assegnare un like.
Elemento tipico: <{(dato), (like), (<amico1,amico2,…>)}>
Struttura dati: private E dato; private int like; private Map<String, Integer> friends;
IV: dato≠null && like≥0 && friends≠null && (∀ amico i ∈ friends => i≠null) && (∀ amico i, j ∈ friends : i≠j => friends[i]≠friends[j])
FA: fa(dato, like, friends)= <{(dato), (like), (<friends.get(key1),friends.get(key2),…>)}>

2.3.2 Category

Overview: tipo di dato modificabile che rappresenta una categoria, i suoi dati e la lista di contatti (amici) a cui saranno visibili i dati.
Elemento tipico: <{ (categoria),(<dato0,dato1,dato2, …>),(<amico0,amico1,amico2, …>) }>
Struttura dati: private String Categoria; private ArrayList<E> Dati; private ArrayList<String> Friends;
IV: Categoria≠null && Dati≠null && Friends≠null && (∀ dato i ∈ Dati => i≠null) && (∀ amico i ∈ Friends => i≠null) && (∀ amico i, j ∈ Friends : i≠j => Friends[i]≠Friends[j])
FA: fa(Categoria, Dati, Friends)=
{(Categoria.nome),(<Friends.get[0],...,Friends.get[size]>),(<Dati.get[0],…,Dati.get[size]>)}

2.3.3 DataBoard

Overview: tipo di dato modificabile che rappresenta una bacheca costituita da una collezione di oggetti generici che estendono il tipo di dato Data. Il proprietario della bacheca può definire le proprie categorie e stilare una lista di contatti (amici) a cui saranno visibili i dati per ogni tipologia di categoria. I dati condivisi sono visibili solamente in lettura: gli amici possono visualizzare il dato ma possono essere modificati solamente dal proprietario della bacheca. Gli amici possono associare un "like" al contenuto condiviso.
Elemento tipico: Sia <{password}, {Categoria1, Categoria2, ...}> un insieme di dimensione arbitraria. L'insieme indica l'insieme delle categorie presenti
nella collezione ognuna identificata da un indice.
Ogni Categoria della collezione ha un elemento tipico definito in questo modo: <{(categoria), (<Dato0,Dato1,Dato2, …>),(<amico0,amico1,amico2, …>) }>.

A sua volta ogni Dato della collezione ha un elemento tipico definito in questo modo:
<{(dato), (like), (<amico1,amico2,…>)}>.
Per Invariante di Rappresentazione e Funzione di astrazione guardare i file Board.java relativi alle due implementazioni.