Relazione Progetto Java PR 2

Venturi Ludovico
Docente: Francesca Levi

UNIPI, Novembre 2019

Indice

1 Scelte progettuali

Nella relazione verranno spiegate le scelte progettuali e implementative che sono state prese.

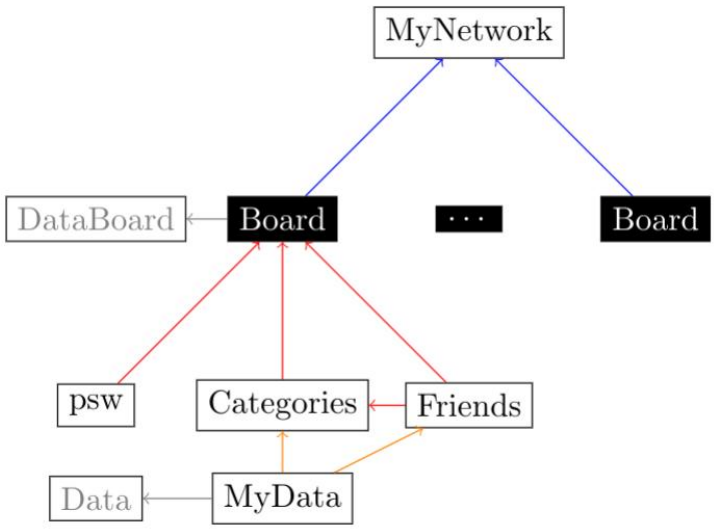

Figura 1: Struttura generale del progetto comune ad entrambe le implemntazioni

1.1 Data

A grandi linee è stata creata una struttura generale dei dati in grado di adattarsi a varie situazioni.
Data viene implementata come *interfaccia*: stabilisce un *contratto* con le sottoclassi che la implementeranno, ovvero:

OVERVIEW: *Data rappresenta un dato astratto sottoforma di un insieme di 2 attributi e alcune operazioni. È una struttura astratta immutable, di dimensione finita e fissa*

Stabilisce delle caratteristiche comuni a tutte le sottoclassi: metodi che dovranno necessariamente implementare:

public void ***display****();*
public String ***getDataTitle****();*
public String ***getCategory****();*

1.1.1 MyData

Viene poi implementata una sottoclasse *astratta* ***MyData*** per soddisfare l'obiettivo iniziale di progettare una struttura versatile.
MyData è implementata come classe astratta e tale scelta deriva dalla volontà

di attribuire a tutte le classi che discendono da *MyData* delle altre caratteristiche comuni, più *concrete*, ovvero dei metodi già implementati e una struttura implementativa di base:

private String ***dataName****;*
private String ***category****;*

Come da specifica la classe *MyData* riporta anche il metodo display , ma astratto: verrà implementato dalle sottoclassi. Da *MyData* discendono i dati veri e propri che saranno salvati in Bacheca: nell'esempio per il progetto sono *Testo* e *Audio*, che:

- ridefiniscono *equals()* per permettere la deep equality
- aggiungono un contenuto
- implementano display()

1.1.2 Ipotesi

- La bacheca risulta una collezione di oggetti di vario tipo, come riportato nel testo, ma ciò non è collegato ad ≪E extends Data≫. Si è pertanto deciso di riportare anche la classe (magari apparentemente superflua) *MyData* in modo da far trasparire chiaramente la presa di coscienza di ciò. La bacheca gestisce dati di tipo E : pertanto ogni dato sottotipo di E, con E definito come ≪E extends Data≫, è un dato valido da inserire in una bacheca basata sul tipo E.
- Non ci sono setter poichè si è ipotizzato che *Data* fosse una struttura *immutable*.

(Nel testo viene riportato *≪i dati possono essere visualizzati dagli amici ma modificati solamente dal proprietario della bacheca≫*: ciò è stato interpretato come: "la modifica consiste nell'aggiunta o la rimozione dei dati, non nella modifica effettiva del contenuto dei dati").

1.2 Board≪E extends Data≫

Board rappresenta un contenitore di oggetti generici. È basata sul tipo generico ≪E extends Data≫ e funzionalmente è una collezione di dati che possono essere di vario tipo, a patto che siano sottotipi di E (che estende *Data*).

Non ho riportato la specifica di ogni metodo nel codice di *Board(2)≪E extends Data≫* poichè risultava troppo confusionario; l'implementazione ha comunque seguito di pari passo la specifica riportata nell'interfaccia *DataBoard≪E extends Data≫*.

1.2.1 Ipotesi

- Non sono ammessi elementi *null*
- Non sono ammessi duplicati di alcun genere

- il numero di likes non dipende solamente dal dato ma anche dalla bacheca in cui si trova ⇒ *Data* non possiede il contatore dei like: questo si trova nella bacheca, relativamente ad ogni dato
- la lista ritornata da *getDataCategory* è in sola lettura, avendo precedentemente ipotizzato che Data sia immutable
- la rimozione di una categoria comporta la perdita dei dati in quella categoria e l'associazione per ogni amico di quella categoria
- i likes sono univoci per ogni amico

Nota sugli Iteratori daie

1.2.2 Implementazione 1

1.2.3 Implementazione 2

2 Eseguire il codice

2.1 Test ed esempi

Non saranno verificate *tutte* i casi in cui parametri sono null per ovvie ragioni, così come tutte le eccezioni ripetute, quali i controlli che la categoria esista o che l'amico sia presente nella lista amici; per queste ultime, pur se generabili in differenti metodi, verranno esplicitamente testate solamente una volta ciascuna. In generale più istruzioni su un singolo blocco *try* indicano che l'ultima sarà quella che genera l'eccezione mentre le precedenti sono eseguite con successo. Lista di test effettuati nel *main*:

- password della bacheca $\ll 8$ caratteri
- get di una bacheca non presente
- password errata
- stringa nulla passata (vale per tutti i metodi)
- categoria già presente
- rimozione di una categoria non presente
- condivisione di una stessa categoria con uno stesso amico
- condivisione di una categoria non presente nella bacheca
- rimozione di un amico non presente nella lista amici
- rimozione di un amico da una categoria cui non ha accesso, anche se presente nella lista amici
- inserimento di un dato già presente
- inserimento di un dato la cui categoria non è presente in bacheca
- get di un dato la cui categoria non è presente
- get di un dato non presente
- get di un dato precedentemente inserito in modo corretto ma la cui categoria è stato poi rimossa
- rimozione di un dato non presente
- getDataCategory e modifica della lista ritornata
- amico vuole inserire like ad un dato di una categoria non condivisa con lui
- amico vuole inserire like ad un dato cui lo ha già messo
- amico vuole inserire like ad un dato non presente
- ITERATORI, prove varie
- elimino una categoria e itero sui dati di tale categoria tramite una amico con cui essa era condivisa