Relazione Sul Progetto Di Java

Oleksiy Nedobiychuk

30 novembre 2020

Indice

1 Introduzione

In questo PDF ho cercato d'illustrare e spiegare le mie scelte ,al meglio delle mie capacita, relative al progetto assegnato. Il file da me inviato si struttura in due classi principali; una chiamata *buildListOfPost* che ha l'obbiettivo basilare di avvolgere tutti i metodi, e variabili d'istanza. Può essere usata come una struttura dati a parte per lavorare sulle liste di tipo *Post*. L'altra classe chiamata *SocialNetwork* ha il compito di presentare l'interfaccia per le manipolazioni sulle classi: *buildListOfPost*, *Map<String, List<Seguire>>*, dove Seguire ha l'intento di tradurre il concetto di "seguire il post", ed infine *Map<String, List<Seguire>>*. I miei propositi erano di scrivere un codice che funzionasse in tutte le situazioni e rispettasse l'invariante di rappresentazione; credo di aver raggiunto il mio scopo.
"Il progetto ha l'obiettivo di applicare i concetti e le tecniche diprogrammazione *Object-Oriented*" Cit. Ho cercato di applicare il concetto di astrazione nascondendo e riusando il codice offerto dalle librerie e scritto da me, e ho implementato tre interfacce. Ho usufruito moltissimo del principio del polimorfismo come ad esempio overriding toString , ma anche con incapsulamento mi sono comportato bene. Ogni metodo che restituisce al chiamante un tipo di dato che potrebbe permettere la modifica dell'informazione attraverso aliasing viene risolta per mezzo di una copia o ritorna un risultato della tipologia non

mutabile.
Per concludere l'introduzione voglio informare che ogni punto del progetto ha una sezione dove cerco di descrivere le mie decisioni e motivarle.

2 Parte 1: Tipo Di Dato *Post*

La mia decisione di fare il tipo di dato *Post* non modificabile è derivata dal concetto che l'utente come figura è assente nella rete sociale. Il codice deve solo rispondere alle interrogazioni fatte sul sistema con dati già presenti dal "cliente". Per semplificare il codice e renderlo più astratto ho deciso d'implementare prima l'interfaccia del tipo, poi farlo diventare comparabile anche se non lo uso esplicitamente nelle classi, ma voglio che ci sia. Per rendere un po più verosimile ho limitato i caratteri che un username può avere, per esempio non può cominciare con un numero o simbolo ma con una lettera seguita da altere lettere, e/o numeri, e/o _ in mezzo. L'invariante di rappresentazione viene rispettato a ogni chiamata al costruttore attraverso il controllo dell'input ed eccezioni.
La mia interpretazione era quella che la Lista aveva bisogno di una classe wrapper ma adesso pensandoci sopra posso dire che si poteva fare a meno della *buildListOfPost* tuttavia ci permette di utilizza il polimorfismo e delegare compiti da *SocialNetwork*, pertanto la mia decisione di tenerla comunque la reputo corretta. Con l'estensione della classe non la voglio sostituire ma solo usufruire dei contenuti presenti nella super classe. Ci permette anche di separare due concetti e implementazioni, inoltre il "cliente" immaginario avrebbe la possibilità di esaminare liste di post di conseguenza è un win win. Tutti i metodi sul PDF del progetto ho deciso di fagli statici per avere delle funzioni pure. Un altro motivo è quello come detto sopra ho interpretato il progetto in modo che il post e social dovessero essere implementati nelle classi diverse, perciò non volevo ripetere per ogni classe lo stesso metodo.

3 Parte 2: Tipo Di Dato *Social Network*

La Social Network unisce le tre componenti creati, la lista dei post, gli utenti che seguono altri utenti , e le persone che seguono i post . Ho deciso di estendere la *buildListOfPost* perché in quel momento era la cosa più sensata, mi servivano metodi che avevo già dichiarati e implementati prima.
La mia principale preoccupazione durante la progettazione di una soluzione era quella di creare metodi efficienti nella gestione del input con meno iterazioni possibili per arrivare a ritornare il risultato, un altro timore era quella di non esporre dati modificabili. Ho creato due generici copyMap e copyOut per copiare le classi che potrebbero aver permesso la modifica non autorizzata. Cosi facendo l'invariante di rappresentazione viene sempre rispettato per tutti

i tipi di dato che si trovano all'interno del Social Network. Ho optato d'implementare il concetto di seguire il post con la creazione del nuovo dato con la sua interfaccia anch'esso immutabile perché sarebbe stato poco pratico avere la classe modificabile di quel tipo, non sarei stato in grado di creare una social in maniera efficace. Per seguire un post basta chiamare uno dei tre metodi, essi prendono o un ID del *Post*, o un *Post* o un testo del *Post* ma segue il primo trovato che sia uguale al quello dato. Un utente non può seguire uno stato due volte, se uno dei metodi viene chiamato con l'username che fa parte delle persone che sono interessati al post, il metodo ritorna la stringa che dice "Stai gia seguendo il *Post*". Un altro metodo interessante è insert che prima di invocare super.insert inserisce e aggiorna la map delle persone che l'username segue. Ho deciso d'interpretare un menzionato come seguito, e se sei stato taggato allora fai sicuramente parte della rete sociale, perciò la variabile che contiene user –¿ seguiti può contenere users che non seguono nessuno. Con il metodo usernameDelete si possono eliminare sia dalla lista che dai post seguiti e utenti seguiti quindi anche le due map potrebbero essere vuote.

4 Parte 3: Estensione Social Network

Prima di rispondere alla domanda vorrei precisare che la mia soluzione presentata non estende la Social Network perché per il principio di LSP dovessi aver dato alla mia sub class tutte le proprietà della super classe, e quindi si sarebbero aggiunti più o meno una centinaia di linee di codice in più, già il mio poggetto è piuttosto lungo, perciò ho scelto d'implementare una soluzione all'interno della social. Ho risolto nel modo seguente: preso un insieme di parolacce in inglese e in italiano, inserito in un TreeMap con le parole come key. Ogni volta che si chiama il metodo insert, la stringa da inserire viene spezzata con l'aiuto di regex in substrings e per ogni parola si va alla ricerca nella Tree, se trova qualcosa allora è sicuramente offensivo. Come detto prima se avessi risolto l'esercizio con una estensione della Social Network alla estesa avrei fatto rispettare il principio di LSP ma la soluzione sarebbe stata simile. Per cercare contenuto offensivo già presente nella lista avrei usato il metodo containing implementato come statico