Relazione progetto Java Programmazione II

Anno Accademico 2020/2021

Studente Luca Rizzo

Matricola 598992

Prefazione

Il processo di analisi delle varie classi da implementare segue una scaletta definita:

1) Descrizione delle variabili di istanza
2) Definizione funzione di astrazione
3) Assunzioni rispettate nell'implementazione dei metodi (IR): risulta complicata la lettura di queste formule dal file java, quindi le assunzioni rilevanti sono riportate con un linguaggio informale per comprendere meglio le scelte dietro l'implementazione
4) L'analisi dei metodi: essa consiste nell'esaminazione di ogni metodo e nella verifica che questi preservino, una volta evocati, l'IR e rispettino la loro specifica, ovvero facciano ciò che la specifica descrive.
   Una classe è ben implementata se ogni metodo rispetta semanticamente la sua specifica e se rispetta il principio di induzione sui dati: dimostriamo che il costrutto restituisce un oggetto che rispetta IR e successivamente esaminiamo ogni metodo e verifichiamo se una chiamata del metodo su un oggetto valido restituisce un oggetto valido.
   In tal modo tutti gli oggetti di quella classe, manipolati con operazioni definite nella classe, saranno oggetti validi.

Eccezioni utilizzate

Oltre alle NullPointerException() ho preferito definirmi due eccezioni PostException(), SocialException() rispettivamente per il tipo di dato Post e MicroBlog, passando come argomento all'oggetto di tipo eccezione creato una stringa di descrizione del perché l'eccezione è stata sollevata. In questo modo evito di definirmi un numero di eccezioni troppo elevato ma riesco comunque, una volta sollevata un'eccezione, a individuare rapidamente il problema e, in caso, a risolvere il codice che ne è la causa: questo meccanismo è stato utilissimo in fase di debug e in fase di test. Nei test ho cercato di controllare tutti i casi "limite" in cui il programma avrebbe potuto funzionare non correttamente e eccezioni di questo tipo mi guidavano nella risoluzione dei problemi e nella verifica di determinati comportamenti.

PARTE 1: Implementazione del tipo di dato Post

Il tipo Post è descritto da 5 variabili d'istanza:

- Id univoco del post : intero univoco che identifica il post e che verrà anche utilizzato nell'implementazione di equals: visto che l'id è univoco, possiamo considerare due post uguali se hanno lo stesso id
- Author: autore del post
- Text: testo del post che non deve superare i 140 caratteri
- Timestamp: indica la data e l'ora di creazione del post
- Likes: set contente gli utenti che hanno messo like al post

E' stato necessario l'uso di un'ulteriore variabile di lavoro, idPost, che sarà utilizzata per assegnare un id univoco a ciascun post. La variabile è dichiarata globale così da incrementare il suo valore come contatore alla creazione di ogni post valido e generare in questo modo id univoci.

Funzione di astrazione

La nostra rappresentazione concreta non differisce dalla rappresentazione astratta: la funzione di astrazione è la funzione identità.

Se un elemento tipico nella sua rappresentazione astratta è indicato con una quintupla,

(id_post, author_post, text_post, timestamp_post,like_post), la funzione d'astrazione applicata ad una rappresentazione concreta restituisce una quintupla contenente le variabili di istanza dell'oggetto:

AF(c) = (id,author,text,timestamp,likes)

Invariante di rappresentazione

Oltre ad imporre che tutti le varabili di istanza siano diverse da null, le assunzioni fondamentali fatte sono:

- Il testo non deve superare i 140 caratteri e non deve essere testo vuoto
- L'autore del post e gli utenti che hanno messo like al post non possono essere la stringa vuota
- Un utente non può mettere like ad un suo post

Tutti gli oggetti validi della classe Post soddisfano queste assunzioni.

Analisi dei metodi e controllo correttezza

Il costruttore istanzia un oggetto di tipo Post con id univoco se i parametri soddisfano i vincoli imposti dall'IR. Altri metodi (*getId(), getText() ,getAuthor() ,toString())* permettono di prelevare informazioni riguardo al post e, una volta eseguiti, non alterano la validità dell'IR (i cosiddetti osservatori). Particolare attenzione però al metodo *getLikes()*: una volta eseguito non può restituire direttamente l'array *likes* perché questo esporrebbe la rappresentazione, consentendo modifiche della variabile d'istanza con metodi non definiti dalla classe. Facendo una deep copy evito il problema dell'aliasing.

Il modificatore *addLikes()*, una volta evocato su oggetti che rispettano IR, continua a rispettare l'IR poiché aggiunge il like al post se e solo se il parametro passato è diverso da null, non è una stringa vuota e il like non proviene dall'autore del post (il metodo che prende una lista di likes esegue questo controllo su tutti gli utenti che hanno messo like).

Il modificatore *deleteLike()*, una volta evocato su oggetti che rispettano IR, continua a rispettare l'IR poiché semplicemente rimuove il like al post se l'utente username aveva messo like al post.

Quindi, il costruttore crea un oggetto Post valido e tutti i metodi che chiamiamo preservano la validità dell'IR. La classe è dunque implementata correttamente.

PARTE 2: Implementazione del tipo Microblog

MicroBlog implementa l'interfaccia SocialNetwork. Per farlo utilizza 3 Map come variabili di istanza:

- *Map<String, Set<String>> utentiSeguiti*: usata per tenere traccia delle corrispondenze fra utente e Set di utenti che egli segue.
  Questa Map definisce anche la semantica di utente iscritto: infatti un utente è iscritto al nostro microblog ⇔ è presente come chiave all'interno di questa Map.
- *Map<String, Set<Post>> postUtente*: usata per tenere traccia della corrispondenza fra utente e post scritti da quell'utente. Se l'utente non ha mai scritto un post non sarà presente in questa Map.
  Un utente può quindi essere iscritto (essere presente come chiave nella Map utentiSeguiti) ma non avere mai pubblicato un post.
- *Map<Integer,Post> postGlobali*: tiene traccia della corrispondenza fra id univoco e post identificato da quell'id. Questa struttura non è necessaria all'implementazione ma il suo utilizzo sarà fondamentale per ridurre la complessità delle operazioni di ricerca di un post nel nostro Microblog. (es. mettere like ad un post identificato da id)

Funzione di astrazione

Il tipo di dato astratto Microblog può essere visto come una funzione che associa ad ogni utente registrato l'insieme degli utenti che egli segue e l'insieme dei post che egli ha scritto.
Il dominio della funzione è l'insieme degli utenti iscritti il codominio è una coppia di insiemi, il primo dei quali è un sottoinsieme degli utenti iscritti (utenti che egli segue) e il secondo è un sottoinsieme di tutti i post pubblicati (solo quelli scritti dall'autore).
Per definire le relazioni di "utente_u segue utente_v" sfruttiamo la nozione stessa di post: tale relazione è intrinseca nel post stesso, ovvero può essere determinata a partire da un insieme di post.
Un utente_u seguirà un altro utente_v ⇔ utente_u ha messo like ad uno dei post di utente_v.
La rappresentazione astratta del tipo di dato MicroBlog può essere quindi ottenuta a partire da un insieme di post.
La nostra funzione di astrazione quindi riesce a generare dallo stato concreto (Map post Utente) lo stato astratto definito come funzione che associa ad ogni utente registrato l'insieme degli utenti che egli segue e l'insieme dei post che egli ha scritto.
La funzione di astrazione mappa per ogni utente_u registrato la lista di utenti che egli segue grazie alla Map utentiSeguiti e i post che egli ha scritto grazie alla Map postUtente:
la Map utentiSeguiti è univocamente determinata dalla Map postUtente, ovvero utente_u segue utente_v <=> ∃ post in postUtente.get(utente) && post.getLikes.contain(user_u).

Meccaniche di funzionamento del tipo di dato e invariante di rappresentazione

Alla base dell'implementazione ci stanno delle scelte effettuate in partenza:

- Un utente è registrato ⇔ è presente come chiave nella Map utentiSeguiti : in tal senso l'insieme degli utenti registrati è ottenibile dal keySet della Map.
- Solo un utente registrato può scrivere un post: di conseguenza tutti i post della rete sono scritti da utenti registrati e posso mettere like e seguire solo utenti registrati.
- Un utente può essere registrato ma non aver mai scritto un post e quindi non essere presente nella Map postUtente: in tal senso tutti gli utenti che sono presenti come chiave in postUtente sono anche presenti come chiave in utentiSeguiti ma non viceversa.
- I likes ai post della rete sono tutti di utenti registrati perché definiscono la semantica del "seguire": tutti gli utenti che hanno messo like ad un post conterranno nella Map followers (quindi sono utenti iscritti), nel corrispettivo set associato all'username dell'utente, l'autore del post al quale hanno messo like.
- Quando si rimuove un like si elimina la relazione che scaturisce da quel like: di conseguenza se utente_u rimuove il like ad un post di utente_v e il like rimosso era l'unico like che utente_u aveva messo ad un post di utente_v, utente_u non seguirà più utente_v.
  Al contrario la relazione di utente_u segue utente_v viene mantenuta perché vi è un altro like di utente_u a un post di utente_v che determina tale relazione (fondamentale per rispettare assunzione di sopra)
- Quando si rimuove un post si eliminano tutte le relazioni che scaturiscono da quel post: è come se ogni utente rimuovesse il like a quel post e di conseguenza la relazione che derivava da quel post. Se quello era l'unico post dell'autore a cui utente_u aveva messo like, elimineremo l'autore del post dagli utenti seguiti da utente_u.

Queste assunzioni saranno alla base della nostra invariante di rappresentazione e del funzionamento di tutti i metodi del tipo di dato MicroBlog.

Analisi dei metodi e controllo correttezza

Le specifiche dei metodi sono realizzate considerando la loro applicazione sulla rappresentazione astratta: in tal senso non saranno inserite nelle clausule effects gli effetti che hanno su strutture di supporto all'implementazione, ovvero quelle strutture concrete che non concorrono alla definizione astratta dell'oggetto rappresentato (es. PostGlobali)

Il costruttore istanzia un oggetto con tre Map vuote che ovviamente rispetta l'IR.

Tra i metodi da implementare, 4 erano metodi statici: *guessFollowers(), influencer(), writtenBy() e getMentionedUser()* non agiscono sulla rappresentazione del dato e quindi preservano l'invariante di rappresentazione. La correttezza nell'implementazione dei metodi scaturisce solamente dal rispetto del comportamento definito della loro specifica: la batteria di test, a questo scopo, verifica il corretto funzionamento dei metodi.

Il metodo *guessFollowers(*) usa la logica e le assunzioni fatte nell'implementazione di Microblog per creare una rete sociale a partire da un insieme di post: questa è un'ulteriore dimostrazione del fatto che la rappresentazione astratta del tipo di dato MicroBlog dipende solo e soltanto dalla lista dei post scritti.

I metodi *writtenBy()* e *getMentionedUser()* non statici e il metodo *containing()* sono dei semplici osservatori che, in tal senso, non modificano la rappresentazione. Anche per loro vale il discorso fatto sopra, con la sola aggiunta che il metodo *writtenBy()* deve fare una "deep copy" dei post che restituisce nella lista poiché, altrimenti, esporremmo la rappresentazione: in questo modo essa potrebbe essere manipolata da metodi non appartenenti alla classe con la possibilità di invalidare l'IR.

In questo modo i post restituisti saranno diversi: avranno id univoco diverso e i vari campi saranno delle copie dei post presenti nella rete.

Il metodo *utentiRegistrati()* restituisce la lista di utenti registrati alla rete. E' un semplice osservatore e valgono i discorsi fatti per i metodi appena descritti.

Meritevoli di maggiori attenzioni sono i modificatori, ovvero i metodi che permettono di alterare la rappresentazione: *loadPost(), likeIt(), createPost(), registraUtente(), removePost(), removeLike() .*

Questi metodi, per essere implementati correttamente, non solo devono rispettare il comportamento definito dalla specifica, ma una volta chiamati su un oggetto che soddisfa l'IR deve preservare l'IR.

Per un'analisi più approfondita si rinvia al pdf in allegato "Analisi Modificatori".

Il tipo MicroBlog è implementato correttamente poiché il costruttore restituisce un oggetto che rispetta IR e tutti i metodi, se applicati su oggetti validi preservano IR. Il comportamento corretto dei metodi, ovvero il rispetto della semantica definita nella specifica è garantito dalla batteria dei test.

PARTE 3: Implementazione estensione gerarchica: MicroBlogFiltrato

Una possibile estensione gerarchica del tipo di dato MicroBlog è rappresentata dalla classe MicroBlogFiltrato di cui è fornita una possibile implementazione.

Il meccanismo che ci sta alla base è quello di passare una lista di parole ritenute offensive e filtrare il testo di ogni post prima che venga pubblicato. Se il post presenta parole offensive questo viene comunque pubblicato con le parole sostituite da asterischi. (Se non avessimo pubblicato il post in caso di presenza di parole offensive questa estensione non avrebbe rispettato il principio di sostituzione).

L'estensione implementata rispetta il principio di sostituzione perché:

- Rispetta la "Signature Rule": gli unici metodi re-implementati (cretePost() e loadPost) hanno una signature compatibile con quella dl padre
- Rispetta la "Properties Rule": gli unici metodi re-implementati (cretePost() e loadPost) non alterano l'invariante di rappresentazione del padre. Le proprietà valide per il super-tipo sono ancora valide per il sotto-tipo
- Rispetta la "Method Rule": gli unici metodi re-implementati (cretePost() e loadPost) mantengono inalterata la precondizione e rafforzano la post-condizione.
  In tal senso questi metodi fanno tutto ciò che i metodi del super-tipo supponevano di fare (pubblicare il post) ma aggiungono qualche effetto in più (censurare parole offensive).
  Ovvero le seguenti regole sono delle tautologie:
  - Precondition Rule : $pre_{super} == pre_{sub}$

- PostCondition Rule : (pre_{super} && $post_{sub}$) => $post_{super}$

Altre possibili estensioni della classe Microblog

Altre possibili implementazioni potrebbero prevedere la possibilità di segnalare un post attraverso un metodo *segnala()*. Quando un post raggiunge un numero limite di segnalazioni verrà eliminato.

Richiede un'ulteriore *Map<int,Set<String>> segnalazioni* che lega ogni post con gli utenti che hanno segnalato il post.

Questa possibile implementazione richiede di implementare un solo nuovo metodo *segnala()* che permette la segnalazione e l'eventuale cancellazione del post se il numero di segnalazione ha superato il numero limite di segnalazioni consentite (Es. se il massimo numero di segnalazioni consentite è k, il post è rimosso ⇔ segnalazione.get(id_post_segnalato).size >= k) . Questo nuovo tipo rispetterebbe le 3 regole definite dalla Liskov nel principio di sostituzione.