Primo progetto intermedio
Programmazione 2
A.A. 2019/2020

Francesco Amodeo
Matricola: 560628
Corso A, prof. Ferrari

Il progetto consiste nella realizzazione dell'ADT `DataBoard<E extends Data>` definito tramite una interfaccia in cui si includono l'overview, il typical element e la specifica formale di tutti i metodi pubblici che il tipo di dato deve offrire.
Oltre alle operazioni richieste sono state aggiunte altre utili operazioni come `toString, displayCategories, displayBoard, repost, getLikes, removeLike, friendAllowed, getFriendCategories`, le cui spiegazioni sul comportamento di tali metodi si rimandano alla specifica.
Per quest'ultima è incluso il javadoc contenente le pagine html che descrivono la specifica e la gerarchia delle classi. La specifica dei metodi è stata scritta usando la sintassi javadoc, ma includendo i custom tags `@requires, @modifies` ed `@effects` utilizzati abitualmente nel corso di PR2.

La classe `Data`

La collezione `DataBoard` è un contenitore di oggetti di tipo generico che estendono il tipo di dato `Data`.
`Data` è un tipo di dato astratto che rappresenta un generico dato e le sue più essenziali proprietà, le quali sono implementate con tre stringhe `private String author,private String name, private String format` (i cui nomi sono auto esplicativi), e con un campo che rappresenta la data di creazione del dato, per il quale è stato usato il tipo `Calendar` della libreria `java.util`.
Nel seguito non vengono usate librerie esterne alla `java.util` ad eccezione della libreria `java.text.SimpleDateFormat` che viene utilizzata in relazione proprio alla formattazione e visualizzazione a video di quest'ultimo campo.
`Data` è un tipo astratto perché oltre ad offrire operazioni di manipolazione e reperimento dei suoi attributi introduce due metodi astratti, `public abstract void display()` e `public abstract Data clone()`, rimandando a estensioni future l'implementazione di tali metodi.
Per scopi dimostrativi sono state definite due classi driver concrete che estendono `Data` (e quindi dovranno obbligatoriamente implementare i suddetti metodi astratti) che fungeranno da contenuto nella collezione `DataBoard`.

Le classi `Text` e `Image`

La prima classe driver è `Text`, tipo di dato che simula un testo e le sue proprietà (`private String text, private String font, private int fontSize, private String color, private boolean bold, private boolean italic, private boolean underline`).
Quest'ultima implementa oltre ai vari metodi getters e setters, i metodi astratti ereditati da `Data`; il metodo `display` è stato implementato stampando la stringa ottenuta dal metodo `toString()` dell'oggetto stesso, mentre il metodo `clone` è stato implementato chiamando il copy constructor definito nella classe `Text`.

La seconda classe è la classe `Image`, tipo di dato che simula una immagine e le sue proprietà (`private String imageText, private int height, private int width`, dove `imageText` è una stringa che simula una immagine costruendola tramite simboli `ASCII`).
Alternativamente, per usi più specifici e non dimostrativi di `DataBoard` contenente immagini, si può ridefinire la classe `Image` utilizzando librerie apposite per la manipolazione reale di immagini come `java.awt.Image`, oppure utilizzare `JavaFX`.
La classe `Image`, come `Text`, estende `Data` e quindi implementa i metodi astratti di quest'ultima nello stesso modo di `Text`.

Le implementazioni di `DataBoard<E extends Data>`

Per il progetto sono richieste due implementazioni dell'interfaccia `DataBoard<E extends Data>` che fanno uso di differenti strutture di supporto.
Nella prima implementazione `FirstDataBoard<E extends Data>` si fa uso della inner class `private class DataOnBoard`, per aggiungere al dato contenuto, le proprietà che lo stesso acquisisce una volta inserito nella bacheca. Questa scelta è stata fatta per ottenere un basso livello di accoppiamento (loose coupling) tra la classe che implementa la bacheca e le classi che definiscono i dati che saranno contenuti nella bacheca. In questo modo si ottiene una facile manutenibilità del software e una ottima indipendenza tra le classi, rendendo più facile anche eventuali estensioni future per tipi di dato, diversi da quelli esemplificativi come `Text` e `Image`, che si desidera inserire nella bacheca.
Nella inner class quindi affianchiamo al campo di tipo generico che indica il tipo del dato da inserire nella bacheca altre proprietà come `private String category, private Calendar postDate, private Set<String> likes` che consistono in categoria a cui appartiene il dato (tra le categorie definite dal proprietario della bacheca), data di pubblicazione del dato nella bacheca(diversa dalla data di creazione del dato) e insieme dei likes ricevuti da parte degli amici che sono autorizzati a visualizzare la categoria del dato e quindi il dato stesso.
La main class fa uso della inner class nella struttura che astrae il contenitore/bacheca per i dati generici, i quali vengono incapsulati nel tipo della `DataOnBoard` acquisendo le sopradette caratteristiche.
Tale struttura è `private Set<DataOnBoard> boardFeed`, un insieme che nel costruttore verrà istanziato nel tipo più specifico `TreeSet`. Nella creazione di questo insieme viene specificato anche un comparatore (tramite espressione lambda), definendo in questo modo l'ordine specifico che si vuole ottenere sull'insieme. Pertanto ogni operazione di aggiunta o rimozione dall'insieme provocherà un riordinamento definito sul campo `postDate` di ogni dato, ottenendo in questo modo una bacheca ordinata dal dato postato più recentemente a quello meno. Il comparatore definisce anche la nozione di uguaglianza, secondo cui due dati vengono considerati uguali se hanno il campo `E data` uguale.
`FirstDataBoard` possiede inoltre una seconda struttura di supporto utilizzata per astrarre una tabella delle categorie definite dal proprietario con associate la lista di amici autorizzati a visualizzare i dati di tale categoria.
Questa struttura è `private Map<String, LinkedHashSet<String>> categoriesDefined`, una mappa che nel costruttore verrà istanziata nel tipo più specifico `Hashmap`.
Le chiavi della mappa sono stringhe di categorie definite dal proprietario della bacheca, mentre i valori associati a tali chiavi sono `LinkedHashSet<String>` insiemi di amici autorizzati a vedere i dati di tale categoria.

La seconda implementazione `SecondDataBoard<E extends Data>`, utilizza direttamente il dato di tipo `E`, senza ulteriori incapsulamenti, ma continua a rispettare gli obiettivi di basso accoppiamento ed estensibilità del software preposti, semplicemente utilizzando ulteriori strutture di supporto per ogni caratteristica che acquisisce il dato una volta immesso nella bacheca.
In particolare, dato che la struttura utilizzata per contenere i dati di tipo `E` è `List<E> boardFeed` vengono utilizzate tante altre liste parallele(`List<Calendar> dataPostDates,List<String> dataCategories, List<List<String>> dataLikes`) quante sono le caratteristiche del dato, mantenendo la corrispondenza con esso tramite le posizione all'interno della lista.
Per implementare l'astrazione della tabella delle categorie definite vengono utilizzate altre due liste in corrispondenza di indici, che sono: `List<String> categoriesDefined` e `List<List<String>> friendsAllowed` dove la prima è la lista delle categorie definite dal proprietario e la seconda è la lista degli amici autorizzati a visualizzare i dati di tale categoria.

Entrambe le implementazioni includono tra i propri campi anche due stringhe che corrispondono al proprietario della bacheca: `String owner` e `String ownerPassw`, username del proprietario e password del proprietario, utilizzata per effettuare controlli di identità sulla operazioni che modificano la bacheca.
Entrambe le implementazioni includono la funzione di astrazione e l'invariante di rappresentazione, dove in quest'ultima sono specificati i vincoli che devono rispettare le due implementazioni. Sono inclusi infatti rigorosi controlli su esistenza e unicità di dati, categorie e utenti, su autorizzazioni a svolgere operazioni e sulla validità dell'inserimento di nuove categorie e nuovi utenti.

La batteria di test

Per valutare il comportamento delle implementazioni proposte è stata realizzata per semplicità una batteria di test statica, che testa tutte le operazioni della bacheca sia in caso di utilizzo corretto da parte dell'utente(rispettando la specifica), sia in caso di utilizzo scorretto. In quest'ultimo caso, dato che il progetto è stato realizzato utilizzando uno stile di programmazione difensiva, ci si aspetta la cattura e il lancio delle eccezioni provocate dall'utilizzo scorretto del dato, con conseguente rifiuto delle richieste dell'utente.