Luca Paulon Matricola: 1187028

Pattern

È stata svolta la prima versione dell'homework.

Le giustificazioni riguardo a che cosa accade ai vari adattatori, se modificati, durante le iterazioni sono state scritte nel documento "Documentazione test" poiché sono state anche testate. Quindi si è data spiegazione dei comportamenti "indefiniti" descrivendo che cosa accade.

Per scelta implementativa nessuno degli adapters sviluppati accetta valori nulli.

Per eseguire il programma bisogna settare da terminale le due variabili di ambiente JUNIT_HOME e CLASSPATH con i valori specificati nel file "Documentazione test" lanciando il terminale dalla cartella in cui è contenuto il file TestRunner.java.

La versione di Junit utilizzata è la 4.12.

Per sviluppare l'homework si sono utilizzati:

- 4 object adapter uno per ognuna delle interfacce da adattare (List, Collection, Set e Map). Si è preferito l'object adapter al class adapter poiché le interfacce da implementare erano molto differenti, in più sia vector che hashtable non hanno metodi che hanno una corrispondenza uno a uno con i metodi delle interfecce target, ma molto spesso si sono utilizzate delle combinazioni di operazioni di vector e hashtable per implementare un metodo della nuova interfaccia. Si hanno quindi 4 sigle way adapter in cui vector e hashtable sono delle istanze di classe.
- Per gli oggetti che implementano l'interfaccia Collection si è utilizzato il pattern iterator. In tutti i casi si è creata una classe interna privata che ritorna gli elementi della collezione secondo un ordine da lei definito. Il vantaggio fondamentale è dato dal fatto che anche degli insiemi di dati che apparentemente non erano iterabili, come hashtable, grazie ad un iteratore lo sono diventati. Questo è stato molto utile per la gestione di KeySet, Values ed EnrtySet che sono dei set e delle collezioni ma con alla base una tabella hash, quindi una struttura dati che intrinsecamente non ha una relazione d'ordine posizionale precedente-successivo come invece ha vector. Lo svantaggio invece è legato al fatto che in alcuni casi la scrittura di un iteratore porta alla duplicazione di codice, poiché tenendo la classe privata interna in alcuni casi molte parti dell'iteratore rimangono invariate ma è necessario che questo conosca la struttura della struttura dati su cui sta operando.
- 3 class adapter, rispettivamente per implementare i metodi keySet(), values() ed entrySet() di Map. In questi casi invece si è preferito implementare dei class adapter perchè derivando KeySet e EntrySet da SetObjectAdapter molti dei metodi dell'adapter base si possono riutilizzare nelle classi derivate e quindi è stata necessaria la sovrascrittura di solo alcuni dei metodi di SetObjectAdapter. Per Values invece dopo alcune valutazioni si è comunque deciso di implementare un class adapter derivando da CollectionObjectAdapter, in questo caso però si è resa necessaria la sovrascrittura della maggior parte dei metodi. Il principale vantaggio è che una Collection ammette duplicati, quindi poiché chiavi distinte possono avere valori uguali nel caso di Values ci possono essere valori duplicati. Lo svantaggio in cui si incorre però è quello di dover sovrascrivere quasi tutti i metodi.
- Infine si è utilizzato un object adapter per la realizzazione dell'oggetto ritornato da sublist. Poiché una sottolista deve avere una lista come backing storage allora si è tenuta una lista di riferimento come istanza di classe e gli indici di inizio e fine della sottolista. Si sono

implementati tutti i metodi dell'interfaccia List attraverso delle combinazioni dei metodi della lista di classe prestando attenzione alla variazione della vista della sottolista. Il vantaggio è stato quello di poter utilizzare i metodi di List già adattati tramite ListObjectAdapter, lo svantaggio quello di dover comunque riadattare l'intera interfaccia List in sublist.