Autore: Fabrizio Cau (508700)
Relazione Progetto Programmazione II 2017/18 Prima Parte (DataCounter)

In questo progetto si è realizzato un tipo di dato modificabile DataCounter che memorizza al suo interno degli elementi unici e ad essi ci associa un valore numerico che ne indica la quantità (il numero di volte che è stato inserito)
Vi sono due classi DataCounter, una implementata utilizzando la classe Hashtable e una seconda mediante struttura intera TreeMap (distinguibili dal nome del file java).
In entrambi i casi si è scelto di istanziare le classi con visibilità private in modo da renderle accessibili direttamente solo dalla classe stessa e non da eventuali sottoclassi o applicazioni ed evitare quindi che vengano effettuate modifiche che invaliderebbero il REP INV.
Il popolamento della collezione viene effettuato tramite il metodo incCount: l'elemento data mandato come parametro viene inserito nella collezione oppure se esiste già viene incrementato il suo valore. Questo metodo è l'unico che può effettuare modifiche alla collezione una volta istanziata.
Tra le eccezioni vi è NullPointerException (presente in java, unchecked), lanciata quando viene passato un parametro null ai metodi incCount e getCount, e EmptyCollectionException (implementata nel file omonimo all'interno del progetto, unchecked) lanciata quando viene chiamato il metodo getIterator su una collezione vuota.

Metodo getIterator:

Il metodo getIterator restituisce un iteratore al chiamante e gli fornisce quindi l'accesso di sola lettura a tutti gli elementi (chiavi) in ordine non crescente per valori e per valori uguali in ordine crescente delle chiavi (il tipo di queste deve quindi estendere comparable). Questo ordinamento non viene dalle classi inizializzate nel costruttore in quanto hashtable non ha un ordine prevedibile di memorizzazione delle chiavi e treemap ordina solo in per ordine naturale delle chiavi, quindi si è scelto di istanziare la classe treeset con un comparatore personalizzato per questo tipo di ordinamento e copiare dentro le chiavi della collezione così che chiamando getIterator da un applicazione avesse un ordine prevedibile come richiesto nella consegna sia istanziando la classe Hashtable che TreeMap.
Per il modo in cui è stato implementato il metodo getIterator in entrambe le classi l'unico modo per vedere eventuali modifiche causate dalla remove sarebbe chiamare getIterator da un applicazione e creare un secondo iteratore copia del primo (evitando di chiamare nuovamente il metodo di DataCounter). Questo perché ogni volta che viene chiamato getIterator viene effettuata un nuova copia della collezione. In ogni caso, per ovviare al problema è stato istanziato un iteratore tramite una classe implementata ad hoc MyIterator (presente all'interno del progetto), che implementa iterator e esegue l'override della remove.

Applicazioni e testing:

Sono state create due applicazioni (ampiamente descritte mediante commenti all'interno della classe stessa): una che popola la collezione con valori interi e un altra, come richiesto dall'esercizio 2 del progetto, che analizza un documento di testo e tiene conto di quante volte viene ripetuta ciascuna parola. In quest'ultimo caso si poteva optare a tenere un applicazione con un codice ridotto e implementare il caricamento del file, la copia e la pulizia della stringa in una classe apposita ma non all'interno di DataCounter in quanto questa classe implementa una struttura di chiavi generiche e dunque non può prevedere che queste siano stringhe.
I documenti di testo utilizzati per testare questa applicazione e le classi DataCounter sono contenuti all'interno della cartella Test.

Conclusioni:

All'esterno della classe, quindi dentro un applicazione che utilizza DataCounter, istanziare la classe con TreeMap o con Hashtable non ha alcuna differenza ne sul popolamento ne sulla visita dei dati inseriti. L'utilizzo di una struttura treemap però prevede la possibilità di tenere gli elementi in un ordine deciso da un comparatore (standard o implementato ad hoc), quindi rispetto alla classe hashtable potrebbe permettere l'implementazione di una collezione di elementi ordinati senza l'uso di altre strutture dati temporanee.