Relazione Homework 2 - v2

Tabella dei contenuti

Design pattern utilizzati

Nello sviluppo del progetto sono stati principalmente due i design pattern applicati tra i diversi pattern della GoF visti a lezione: l'adapter pattern e l'iterator pattern.

L'**adapter pattern** (structural pattern) è stato applicato per permettere di utilizzare una libreria "myLib" nata in un ambiente Java J2SE 1.4.2 in un ambiente più limitato, JME CLDC 1.1. Più nel dettaglio, nella libreria "myLib" erano presenti delle classi che facevano uso di alcune interfacce (Map, List, Set e Collection) disponibili nell'ambiente più avanzato ma non in quello più limitato. È stato quindi necessario definire le interfacce e la loro implementazione, adattando mediante degli adapter il comportamento di alcune strutture dati presenti nella versione CLDC 1.1 alle specifiche definite dalle interfacce del J2 Collection Framework.

Gli adapter sono stati definiti come object adapter, contenenti al loro interno un'istanza dell'oggetto "adattato" (è stata utilizzata quindi la composizione, a differenza invece dei class adapters, basati sull'ereditarietà). È stata scelta questa particolare implementazione del pattern in quanto permette di minimizzare il coupling tra adapter e adaptee e di rendere il funzionamento del sistema più lineare e comprensibile, oltre che aumentare la facilità di test sul sistema stesso. Il principale svantaggio a fronte di questi vantaggi consiste nella necessità di scrivere del codice aggiuntivo per adattare tutte le operazioni sull'adapter in corrette operazioni sull'adaptee (anche se in questo caso, non essendovi una diretta corrispondenza tra le funzionalità dei due, sarebbe stato necessario comunque adattare molti comportamenti anche utilizzando un class adapter).

L'**iterator pattern** (*behavioral pattern)* è stato invece applicato per consentire l'accesso e la navigazione sequenziale delle strutture dati definite dagli adapter senza esporne i dettagli implementativi all'esterno. Grazie all'utilizzo di questo pattern è possibile per gli algoritmi che operano sulle collezioni effettuare un disaccoppiamento tra l'accesso ai dati e l'implementazione concreta delle strutture stesse. Questo rappresenta il principale vantaggio derivante dall'utilizzo di questo pattern, a fronte di un unico svantaggio costituito invece dalla necessità di definire un'ulteriore classe all'interno del proprio codice.

Testing

L'intero progetto è stato sviluppato applicando la metodologia *test-driven development*: per prima cosa sono stati scritti i test di ciascuna classe e solo in seguito sono state implementate le classi stesse, controllando passo passo durante la stesura del codice la correttezza dello stesso. Al termine dell'implementazione i test sono stati riorganizzati e documentati secondo quelle che sono le specifiche definite dal template SAFe di IBM.

I test sono stati implementati utilizzando il framework di testing JUnit (versione 4.13). Oltre alle notazioni @Test, @Before e ai vari assert utilizzati per valutare specifiche condizioni sulle istanze delle classi testate (assertEquals, assertArrayEquals) sono state sfruttate anche la possibilità di parametrizzare le suite di testing e la possibilità di definire un ordine specifico di esecuzione dei test (nel caso solo allo scopo di ordinare l'output dei test nell'IDE).

La parametrizzazione delle test suite ha permesso di ottenere un maggior riutilizzo del codice e una maggiore elasticità delle test suite implementate.

La documentazione dei test è stata redatta, come già specificato precedentemente, secondo il formato SAFe. Al fine di permettere una generazione automatica delle tabelle di documentazione necessarie a partire dalla Javadoc dei singoli metodi è stata definita, insieme al collega di corso Riccardo Forzan, una standardizzazione dei commenti secondo un sistema di tag specifici (link alla repository dello script utilizzato per la generazione della documentazione). Le tabelle di documentazione delle singole test suite sono allegate alla consegna (nella cartella testDoc). Di seguito è invece riportata una breve descrizione di ognuna della test suite definite.

> CollectionTest - Test suite che raccoglie tutti i test definiti al fine di verificare il corretto funzionamento di un'istanza di una classe che implementa l'interfaccia Collection (solo i comportamenti generali definiti da Collection vengono testati in questo caso). La suite è parametrizzata, ovvero gli stessi test sono eseguiti più volte su parametri diversi, definiti dallo sviluppatore (sono state parametrizzate istanze di Set, List e SubList). Questa suite fa uso di Reflections. I test della suite vengono eseguiti in ordine alfabetico crescente. È presente un attributo privato `instance` di tipo HCollection che viene inizializzato con una nuova istanza di oggetto che implementa l'interfaccia prima dell'esecuzione di ogni test.
>
> SetTest - Test suite che raccoglie tutti i test definiti al fine di verificare il corretto funzionamento di un'istanza di una classe che implementa l'interfaccia Set (solo i comportamenti specifici definiti dall'interfaccia Set sono testati; comportamenti più generici, ereditati ad esempio da Collection, sono testati in Collection). Anche in questo caso la suite è parametrizzata (ma vi è un solo parametro, un'istanza di Set) ed i test vengono eseguiti in ordine alfabetico crescente. È presente un attributo privato `instance` di tipo HSet che viene inizializzato con una nuova istanza di oggetto che implementa l'interfaccia prima dell'esecuzione di ogni test.

ListTest - Test suite che raccoglie tutti i test definiti al fine di verificare il corretto funzionamento di un'istanza di una classe che implementa l'interfaccia List (solo i comportamenti specifici di List vengono testati). Anche questa suite è parametrizzata (i test non solo eseguiti solo su di una lista normale ma anche su di una SubList) e i test vengono eseguiti in ordine alfabetico crescente. È presente un attributo privato `instance` di tipo HList che viene inizializzato con una nuova istanza di oggetto che implementa l'interfaccia prima dell'esecuzione di ogni test.

MapTest - Test suite che raccoglie tutti i test definiti al fine di verificare il corretto funzionamento di un'istanza di una classe che implementa l'interfaccia Map e delle istanze di Set e Collection che sono restituite dai suoi metodi. In questa suite i test vengono eseguiti in ordine alfabetico crescente. È presente un attributo privato `instance` di tipo Map che viene inizializzato con una nuova istanza di mappa prima dell'esecuzione di ogni test.

TestRunner - Modulo che avvia l'esecuzione dei test e ne stampa i risultati.

Risultati esecuzione test

```
Running tests ...
____________________

Now executing test: tester.MapTest
Test successful: true.
Test statistics: 104 succeeded, 0 ignored, 0 failed.
____________________

Now executing test: tester.ListTest
Test successful: true.
Test statistics: 142 succeeded, 0 ignored, 0 failed.
____________________

Now executing test: tester.SetTest
Test successful: true.
Test statistics: 14 succeeded, 0 ignored, 0 failed.
____________________

Now executing test: tester.CollectionTest
Test successful: true.
Test statistics: 177 succeeded, 0 ignored, 0 failed.
____________________

Total results: 437 succeeded, 0 failed, 0 ignored.
```