Progetto: Adapter per Java Collections Framework (CLDC 1.1)

Descrizione del Progetto

Questo progetto implementa un Adapter che consente l'utilizzo di interfacce ispirate al Java 2 Collections Framework (versione 1.4.2) in un ambiente Java Micro Edition (Java ME) con configurazione CLDC 1.1. L'obiettivo è rendere utilizzabili le funzionalità di `java.util.List` e delle sue interfacce correlate (`Collection`, `Iterator`, `ListIterator`) in un contesto in cui è disponibile nativamente solo la classe `java.util.Vector` di CLDC 1.1.

Il progetto si basa sul pattern Adapter, utilizzando `java.util.Vector` come "adaptee" per fornire l'implementazione delle funzionalità definite dalle interfacce "target" personalizzate.

Motivazione e Scopo

L'obiettivo principale è superare le limitazioni di compatibilità tra codice Java sviluppato per J2SE (Standard Edition) e ambienti più restrittivi come CLDC 1.1. Questo è cruciale in scenari dove è necessario riutilizzare logica di business esistente basata sul Collections Framework completo, ma l'ambiente target non lo supporta nativamente.

L'adapter fornisce una soluzione permettendo al codice J2SE di interagire con le strutture dati disponibili in CLDC 1.1 attraverso un'interfaccia familiare, evitando una riscrittura completa della logica applicativa.

Architettura e Design

Il progetto è strutturato secondo il **Pattern Adapter** e rispetta le seguenti specifiche:

Package Principale

Tutto il codice dell'adapter risiede nel package `myAdapter`.

Interfacce Target Personalizzate (myAdapter)

Per evitare collisioni e garantire la compatibilità con CLDC 1.1 (che non include le interfacce Collections Framework standard), sono state definite localmente le seguenti interfacce, che replicano fedelmente i metodi delle loro controparti J2SE 1.4.2:

- **HCollection** (replica di `java.util.Collection`)
- **HList** (replica di `java.util.List`, estende `HCollection`)
- **HIterator** (replica di `java.util.Iterator`)
- **HListIterator** (replica di `java.util.ListIterator`, estende `HIterator`)

Classe Adapter Principale (myAdapter.ListAdapter)

- Implementa l'interfaccia `HList` e incapsula un'istanza di `java.util.Vector` (di CLDC 1.1) come adaptee
- Tutti i metodi di `HList` sono implementati delegando le operazioni al Vector sottostante, assicurando la conformità alle specifiche di CLDC 1.1
- Include l'implementazione della sottolista (SubList) come classe interna, che funge da vista coerente sulla ListAdapter genitore
- Da notare che tra i costruttori sono presenti quello di default e quello con un int size della classe vector

Implementazione dell'Iteratore (myAdapter.ListIterator)

- Implementa `HListIterator` e `HIterator`
- Agisce da adapter per traversare e modificare la ListAdapter sottostante, utilizzando le funzionalità di CLDC 1.1

Gestione delle Eccezioni Personalizzate (myExceptions)

Le eccezioni standard di Java SE come `java.lang.UnsupportedOperationException` e `java.lang.IllegalStateException` non sono disponibili in CLDC 1.1. Per superare questa limitazione, sono state create implementazioni personalizzate di queste eccezioni nel package `myExceptions`:

- `myExceptions.UnsupportedOperationException`
- `myExceptions.IllegalStateException`

Queste eccezioni vengono lanciate nei metodi dell'adapter dove previsto dalle specifiche J2SE, garantendo il comportamento atteso anche in un ambiente limitato.

Nota: `java.lang.IndexOutOfBoundsException` è invece disponibile in CLDC 1.1 e viene utilizzata direttamente.

Struttura delle Directory

```
.
├── myAdapter/
│   ├── HCollection.java
│   ├── HList.java
│   ├── HIterator.java
│   ├── HListIterator.java
│   ├── ListAdapter.java
│   └── ListIterator.java
├── myExceptions/
│   ├── IllegalStateException.java
│   └── UnsupportedOperationException.java <-- Aggiunto, se implementato
└── myTest/
    ├── AllTestsSuite.java
    ├── TestListAdapterEmpty.java
    ├── TestListAdapterPopulated.java
    ├── TestListIteratorEmpty.java
    ├── TestListIteratorPopulated.java
    ├── TestSubListAdapter.java
    └── TestRunner.java
```

Come Compilare ed Eseguire

Il progetto è sviluppato in Java e può essere compilato ed eseguito utilizzando un compilatore Java (JDK) e JUnit 4.

Prerequisiti

- Java Development Kit (JDK) 8 o superiore
- JUnit 4.13.2 e Hamcrest 1.3 (i JAR sono forniti nella directory `JUnit/`)

Compilazione

Per compilare il progetto, posizionarsi nella directory radice del progetto (quella che contiene le cartelle `myAdapter`, `myExceptions`, `myTest`) ed eseguire il seguente comando:

Windows:

```
javac -cp "JUnit/junit-4.13.2.jar;JUnit/hamcrest-core-1.3.jar" -d bin myAdapter/*.java myTest/*.java myExceptions/*.java
```

Linux/macOS:

```
javac -cp "JUnit/junit-4.13.2.jar:JUnit/hamcrest-core-1.3.jar" -d bin myAdapter/*.java myTest/*.java myExceptions/*.java
```

> ***Nota:*** *Su sistemi Unix-like (Linux/macOS), il separatore del classpath è `:` anziché `;`.*

Verrà creata una cartella `bin/` contenente tutti i file `.class` compilati.

Esecuzione dei Test

Per eseguire la suite completa di test JUnit, utilizzare il TestRunner fornito. Dopo aver compilato il progetto, eseguire il seguente comando dalla directory radice:

Windows:

```
java -cp "bin;JUnit/junit-4.13.2.jar;JUnit/hamcrest-core-1.3.jar" myTest.TestRunner
```

Linux/macOS:

```
java -cp "bin:JUnit/junit-4.13.2.jar:JUnit/hamcrest-core-1.3.jar" myTest.TestRunner
```

Creazione dei javadoc

Se non si riescono ad aprire i vari javadoc, qui e' presente il comando utilizzabile per poter crearli:

```
javadoc -d doc -cp ".;JUnit\junit-4.13.2.jar;hamcrest-core-1.3.jar" -sourcepath . myAdapter myExceptions myTest
```

Test

Il progetto include una suite di test esaustiva, situata nel package `myTest`, organizzata per coprire i vari aspetti dell'adapter e delle sue interfacce:

Classi di Test

- **TestListAdapterEmpty.java**: Test per ListAdapter su una lista vuota, coprendo i casi limite e la gestione delle eccezioni --> *46 Tests*
- **TestListAdapterPopulated.java**: Test per ListAdapter su una lista popolata, verificando accesso, modifica, ricerca, conversione in array, gestione di elementi null, indici validi/non validi, aggiunta/rimozione multipla, uguaglianza e hashCode. Include numerosi test dettagliati per ogni metodo --> *90 Tests*
- **TestListIteratorEmpty.java**: Test per ListIterator su una lista vuota, focalizzandosi sul comportamento dell'iteratore in assenza di elementi --> *15 Tests*
- **TestListIteratorPopulated.java**: Test per ListIterator su una lista popolata, verificando la navigazione bidirezionale, le operazioni di modifica (add, remove, set) e la gestione dello stato interno --> *28 Tests*
- **TestSubListAdapter.java**: Test dedicati alla classe interna SubList di ListAdapter, assicurando che si comporti come una vista coerente sulla lista genitore e che le modifiche si propaghino correttamente --> *60 Tests*
- **AllTestsSuite.java**: Una suite JUnit che aggrega tutti i test sopra menzionati per un'esecuzione combinata
- **TestRunner.java**: Un'utility per eseguire la AllTestsSuite da riga di comando e presentare i risultati in modo leggibile

Aspetti Coperti dai Test

La progettazione dei test mira a garantire la conformità alle specifiche J2SE 1.4.2 per i comportamenti di List e ListIterator, pur operando su CLDC 1.1. Particolare attenzione è stata data a:

- Gestione degli elementi null
- Comportamento degli indici fuori limite
- Propagazione delle modifiche tra lista e sottolista
- Stato interno di ListIterator (cursore, lastReturned)
- Coerenza di `equals()` e `hashCode()`
- Lancio delle eccezioni appropriate (`IndexOutOfBoundsException`, `NullPointerException`, `myExceptions.IllegalStateException`, `myExceptions.UnsupportedOperationException`, `myExceptions.NoSuchElementException`)

Accorgimenti Specifici

- **Documentazione Javadoc**: È presente una directory `doc/` contenente la documentazione Javadoc generata per tutti i package del progetto
- **Supporto null**: L'implementazione della lista permette il contenuto di oggetti null come elementi, in linea con le specifiche di `java.util.List`
- **Costruttori di Default nei Test**: Alcune classi di test includono costruttori di default espliciti con commenti Javadoc per evitare warning durante la generazione della documentazione Javadoc, pur non essendo strettamente necessari per il funzionamento di JUnit
- **Messaggistica dei Test**: Per scelta progettuale, i test più semplici non producono stampe in console in caso di fallimento, mentre per i test più complessi sono presenti messaggi dettagliati per facilitare il debug

Tecnologie Utilizzate

- **Java**: Linguaggio di programmazione (compatibile con JDK 8+)
- **CLDC 1.1 (Java ME)**: Ambiente di riferimento per le funzionalità di base utilizzate (in particolare `java.util.Vector`)
- **JUnit 4.13.2**: Framework per il testing unitario
- **Hamcrest 1.3**: Libreria di matcher utilizzata con JUnit

Autore

Alberto Bortoletto 2101761