3. Arbeitsblatt: Iteratoren und doppelt verkettete Listen

Sie haben nun die Funktionsweise von Iteratoren kennengelernt. Mit diesem Arbeitsblatt werden Sie das erlernte anwenden, um Schritt für Schritt eigene Iteratoren zu implementieren. Im ersten Teil sollen Sie für Ihre `MyLinkedList` (einfach verkettete Liste aus dem Arbeitsblatt 2) einen `Iterator` bauen. Im zweiten Teil werden Sie eine neue Listen-Variante, eine doppelt verkettete Liste, und einen für Listen spezifischen `ListIterator` implementieren. Dieser erweitert die Funktionalität eines einfachen `Iterators` um zusätzliche Operationen.

Importieren Sie dazu das Programmier-Projekt `ch.fhnw.algd2.iterators`. Beachten Sie folgendes:

- In diesem Projekt gibt es mehrere Testordner. Der Test Source-Ordner `testbase` enthält abstrakten Testklassen, die von den anderen Testklassen benutzt werden und die Sie nicht direkt aufrufen sollten. Weil die Funktionalitäten auf beiden Listen gleich sind, sind die Tests zentral programmiert um dann in den konkreten Test-Klassen verwendet zu werden.

  Wichtig für Sie sind die Testordner `test1sll` (`SinglyLinkedList`-Test) und `test2dll` (`DoublyLinkedList`-Test). Die in den Aufgaben angegebenen Tests beziehen sich immer auf den entsprechenden Test-Ordner.

- Testen Sie jeweils nach der Implementation einer Teilaufgabe, bevor Sie die nächste implementieren. Die Teilaufgaben hängen teilweise voneinander ab, so dass zuerst alle Tests erfolgreich durchlaufen sollten, bevor Sie mit der nächsten Teilaufgabe beginnen.

- Für alle Teilaufgaben existiert eine Klasse mit Unit-Tests. Die Namen der Testklassen werden jeweils in der Teilaufgabe angegeben. Sie können diese Unit-Tests einzeln ausführen, um die entsprechende Teilaufgabe separat zu testen. Es empfiehlt sich jedoch jeweils gleich alle Tests auszuführen (Rechtsklick auf den test-Ordner -> *Run as -> JUnit Test*). Dies hat den Vorteil, dass nur eine Run-Configuration erstellt werden muss. Zusätzlich sehen Sie später gleich, ob alle vorangehenden Tests nach wie vor erfolgreich sind. Fehlgeschlagene Tests späterer Aufgaben können einfach ignoriert werden.

- Falls Sie zwei Implementationen machen für die doppelt verkettete Liste, dann können Sie die Tests für die zweite Implementation ausführen, indem Sie die Klasse `DoublyLinkedListFactory` im Package `ch.fhnw.algd2.collections.list` anpassen (im `test2dll` Source-Paket). Dort kann anstatt der per Default eingestellten `DoublyLinkedList`-Instanz eine Instanz Ihrer anderen Liste zurückgegeben werden.

- Aus den `Iteratoren`-Klassen (innere Element-Klassen) können Sie Methoden der umgebenden `List`-Klasse direkt aufrufen. So können sie Zeit und Code-Duplikation sparen.

1. Implementation des Iterators auf der einfach verketteten Liste

(Die angegebenen Unit-Tests befinden sich alle im Source-Paket `test1sll`)

A. Setup

Sie benötigen eine Implementation der einfach verketteten Liste (`MyLinkedList`) aus dem Arbeitsblatt 2. Voraussetzung ist, dass die Implementation alle Unit-Tests von „A_...“ bis „D_...“ des Arbeitsblatts 2 besteht. (Hinzufügen von `null`-Werten ist nicht erforderlich). Ist das bei Ihrer Implementation (noch) nicht der Fall, verwenden Sie die Musterlösung. Kopieren Sie den Inhalt in die Klasse `SinglyLinkedList`, beachten Sie aber, dass die Methode `iterator()` sowie die komplette innere Klasse `MyIterator` erhalten bleiben!

Führen Sie alle Tests des Pakets `test1sll` (`SinglyLinkedList`-Test) aus. Stellen Sie sicher, dass alle Tests des Pakets `ch.fhnw.algd2.collections.list.linkedlist` erfolgreich sind. Dies ist die Voraussetzung, dass die `Iteratoren` überhaupt korrekt erzeugt und getestet werden können.

B. Implementation eines einfachen Iterators

In der Klasse `SinglyLinkedList` befindet sich die Methode `iterator()`, die einen `Iterator` zurück liefert. Dieser `Iterator` ist eine Instanz der inneren Klasse `MyIterator`. Diese wurde jedoch noch nicht implementiert. Implementieren Sie in einem ersten Schritt die beiden Methoden:

- `hasNext()`
- `next()`

Sie brauchen noch keinen Modification Counter (Generationszähler) zu führen. Ihre Implementierung sollte der Spezifikation des `Iterator`-Interfaces[1] entsprechen. Testen Sie ihre Implementation mit der Unit-Test Klasse `A_SLL_Iterator_SimpleIteration`

C. Führen eines Modification Counter (Generationszähler)

Ändern Sie nun Ihre Implementierung (`Iterator` und Listen-Klasse) so ab, dass Änderungen an der Liste, die nicht vom `Iterator` selbst vorgenommen wurden, erkannt werden. Die Methode `next()` soll eine `ConcurrentModificationException` werfen, falls sich die Liste geändert hat seit das Iterator erzeugt wurde.

Testen Sie Ihre Implementation mit der Unit-Test Klasse `B_SLL_Iterator_CondurrentModification`

D. Implementation der Remove-Methode

Implementieren Sie nun noch die Remove-Methode für den `Iterator` nach der Java-Spezifikation. Testen Sie ihre Implementierung mit dem Unit-Test `C_SLL_Iterator_RemoveElement`

2. Implementation einer doppelt verketteten Liste und eines ListIterator

(Die angegebenen Unit-Tests befinden sich alle im Source-Paket `test2dll`)

A. Implementation einer doppelt verketteten Liste

Implementieren Sie in der Klasse `DoublyLinkedList` eine doppelt verkettete Liste. In dieser Listenstruktur haben die einzelnen Nodes nicht nur einen Zeiger auf das nächste Element (`next`), sondern auch einen auf das vorherige (`prev`). Sie können Ihre einfach verkettete Liste als Grundlage verwenden. Ob Sie nur eine doppelt verkettete Liste oder auch eine Ringliste implementieren ist Ihnen überlassen. Beachten Sie aber: Falls sie die Ringliste implementieren, müssen sie bei der `toArray()`-Methode die Return-Anweisung auf `return arrayForCyclicDoublyLinkedList();` ändern. (Diese Methode ist bereits vorhanden.)

Tipp: Es hilft, die verschiedenen Situationen aufzuzeichnen, um zu sehen, wie die Referenzen jeweils verändert werden müssen und welche Spezialfälle es gibt.

Testen Sie mit den Unit-Tests aus `ch.fhnw.algd2.collections.list.linkedlist` (Tests A_ bis D_).

B. Implementation des Iterators

Implementieren Sie einen einfachen `Iterator` gemäss der Aufgabe 1.B für die Klasse `DoublyLinkedList`. Testen Sie ihre Implementation mit der Unit-Test Klasse `A_DLL_Iterator_SimpleIteration`

C. Implementation der Remove-Methode auf dem Iterator

Implementieren Sie die Methode `remove` (vgl. 1.C) auf dem `Iterator` Ihrer `DoublyLinkedList` Klasse. Testen Sie Ihre Implementation mit der Unit-Test Klasse `B_DLL_Iterator_CondurrentModification`

D. Implementation des ListIterators

Informieren Sie sich über den `ListIterator`[2]. Ändern Sie nun die `implements`-Deklaration Ihres `MyIteratrors` der Klasse `DoublyLinkedList` von `Iterator` nach `ListIterator` und implementieren Sie die zusätzlich erforderlichen Methoden.

Passen Sie die Methode `listIterator()` so an, dass sie keine `UnspportedOperationException` mehr wirft, sondern eine Instanz Ihrer `MyListIterator` Klasse zurückgibt.

Testen Sie Ihre Implementation mit der Unit-Test Klasse `D_DLL_ListIteratorTest.`

[1] Iterator-Interface von Java 11: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Iterator.html

[2] ListIterator von Java 11: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/ListIterator.html