Übung 04: Iteratoren

In dieser Übung implementieren wir Iteratoren für die Datenstrukturen Stack und Set aus den letzten beiden Übungen.

Aufgabe 1: Iterator für Stack

Gegeben ist eine generische Implementierung für das Set, in Anlehnung an die Musterlösung der Übung 3.

- Implementieren Sie die durch das Interface `Iterable` vorgeschriebene Methode `iterator`.
- Implementieren Sie dazu einen Iterator, welcher alle Elemente besucht.
- Stellen Sie sicher, dass der Testcase `TestSet.testStringSet` korrekt durchläuft.

Hinweise

- Die Verwendung einer Agenda ist zwingend notwendig, sie müssen sich also irgendwie merken, was schon ausgegeben wurde. Dazu können Sie z.B. auch den Stack verwenden.
- Was ändert sich an der Besuchsreihenfolge, wenn Sie für die Agenda statt einer Liste einen Stack verwenden?

Aufgabe 2: Reverse-Iterator für `Stack<T>`

Gegeben ist eine generische Implementierung eines Stapels (stack), in Anlehnung an die Musterlösung der Übung 3. Ein Stack ist eine Datenstruktur, dessen Reihenfolge bei der Entnahme (pop) umgekehrt zur Reihenfolge des Hinzufügens (push) ist. Das heißt, das zuletzt mit push hinzugefügte Element das wird als nächstes mit pop entnommen (last in, first out).

- Implementieren Sie (nur) die durch das Interface `Iterable` vorgeschriebene Methode `iterator`. Implementieren Sie dazu einen Reverse-Iterator, der die Elemente in umgekehrter Reihenfolge besucht. Für einen Stack bedeutet das: in der Reihenfolge, in der die Elemente hinzugefügt wurden, also first in, first out.
- Stellen Sie sicher, dass der Testcase `StackTest.testStack` korrekt durchläuft.

Ein Beispiel: Werden in einen Stack<Integer> die Werte 1, 4, 5, 2 gepusht, so ist die Reihenfolge bei pop 2, 5, 4, 1. Iteriert man hingegen darüber, so soll die Einfüge Reihenfolge eingehalten werden und es sollen wieder die Werte 1, 4, 5, 2 ausgegeben werden.

Hinweise

- Auch die Iteration einer sequenziellen Datenstruktur kann mit einer Agenda realisiert werden; wie verhält sich diese über die Lebensdauer (der Iteration)?
- Welche Datenstruktur eignet sich, um die Reihenfolge zu invertieren?

Aufgabe 3: Blattiterator

Implementieren Sie die Methode `SetImpl<T>.leafIterator`.

- Implementieren Sie diesen Iterator so, dass nur Blätter zurückgegeben werden, also Elemente ohne Nachfolger.
- Stellen Sie sicher, dass der Testcase `SetTest.testLeafIterator` korrekt durchläuft.

Hinweise

Diese Aufgabe ist aussergewöhnlich trickreich! Die Hauptschwierigkeit besteht darin, dass auch die `hasNext` Methode verlässlich funktioniert. Eine Herangehensweise ist, neben der Agenda für die Traversierung der Baumstruktur auch eine Referenz auf das nächste Blatt zu unterhalten. Wird nun das nächste Element angefordert, so kann man dieses schnell zurückgeben, muss aber anschließend gleich versuchen, das nächste zu finden.