Übung 8 – Iterationen

Aufgabe 1

Schreiben Sie eine generische Klasse `Iterator1DArray<T>`, die `Iterator<T>` implementiert, für die Iteration über 1D-Arrays. Dieser Iterator soll drei Möglichkeiten bieten vorwärts (niedriger Index zu hohem Index) über ein Array zu iterieren. Standartmäßig soll dieser Iterator vom Anfang bis zum Ende des Arrays iterieren. Außerdem soll es die Möglichkeit geben von einem Index `start`, bis zum Ende des Arrays zu iterieren. Des Weitern soll es die Möglichkeit geben von einem Index `start` bis zu einem Index `ende` (exklusive) zu iterieren. *Da wir hier nichts weiter für* `ende` *spezifiziert haben, zu welchem Problem könnte es kommen?*

Hinweis: Überlegen Sie zunächst wie Sie dies geschickt umsetzen, möglichst ohne zeilenweise ähnlichen/gleichen Code zu schreiben. Es ist hierbei günstig einen Basisfall zu schreiben (welcher ist dies?) und alle anderen darauf zurückzuführen (überladener Konstruktor, `this`). Machen Sie sich ggfs. eine Skizze.

Aufgabe 2

Schreiben Sie eine generische Klasse `SnakeIterator2DArray<T>`, die `Iterator<T>` implementiert. Sie soll Iteratoren darstellen, die ein 2-dimensionales Array von Werten vom Typ `T` jeweils einmal komplett in einer schlangenartigen Weise durchlaufen. Das bedeutet, wenn der Zeilenindex gerade ist werden die Spalten von vorne nach hinten durchlaufen, wenn der Zeilenindex ungerade ist werden die Spalten von hinten nach vorne durchlaufen. Der Iterator erhält bei Erzeugung als Argument eine Referenz auf das zu durchlaufende Array.

Beispiel: Für das 2-dimensionale Array

```
Integer[][] a = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

erzeugt `new Iterator2DArray<Integer>(a)` einen Iterator, der die Werte `1 2 3 4 8 7 6 5 9 10 11 12` liefert.

Aufgabe 3 (Theorie)

Gegeben sei eine Klasse `ForEachWrite`, die eine `main` mit folgendem Codeausschnitt enthält.

```
int[] arr = {1,2,3};
for(int element : arr){
        element = 42;
}
```

Beschreiben Sie mit eigenen Worten was in diesem Codeausschnitt passiert. Was würde passieren, wenn wir statt eines `int[]` ein `Puffer<Integer>` in der erweiterten for-Schleife verwenden würden (`Puffer` sollte `Iterable` erweitern, siehe unten)? Beschreiben Sie auch dies mit eigenen Worten.

Aufgabe 4

Damit unsere ADTs nun auch *alle* iterierbar werden müssen, lassen Sie Ihre Schnittstelle `Puffer` die Schnittstelle `Iterable` erweitern. Machen Sie außerdem die konkreten Klassen `DynArray` und `EVL` iterierbar, indem sie die Schnittstelle `Iterable` implementieren. Implementieren Sie hierzu jeweils einen entsprechenden Iterator als innere private Klasse – `DynArrayIterator` bzw. `EVLIterator`. Die Iteratoren sollen jeweils von Anfang bis zum Ende durch die Datenstrukturen iterieren. Machen Sie zuletzt unsere Wrapper-Klassen Klassen `SchlangeMitEVL` und `FolgeMitDynArray` iterierbar – also lassen Sie diese auch `Iterable` implementieren.