ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Science

Programmierung 2 - Sommersemester 2022

Prof. Dr. Markus Esch

Übungsblatt Nr. 15
Abgabe KW 19

1. Aufgabe
   Erweitern Sie die Klassen `PersonQueue` aus Übungsblatt 10 um einen internen Iterator. Der Iterator soll nur innerhalb der Klasse nutzbar sein und folgendes Interface implementieren:

```
public interface PersonIterator {
  /**
  * Gibt true zurück, wenn der Iterator weitere Elemente enthält.
  * Ansonsten false.
  * @return true, wenn der Iterator weitere Elemente enthält.
  */
  public boolean hasNext();
  /**
  * Gibt das nächste Person-Objekt zurück
  * @return das nächste Person-Objekt.
  */
  public Person next();
}
```

   Erweitern Sie die Klasse `PersonQueue` um eine Methode `void print()`. Diese soll die gesamte Queue mithilfe des Iterators ausgeben. Erweitern Sie die Klasse `PersonQueue` um eine weitere Methode `String smallest()`. Diese soll die Person in der Warteschlange mit dem lexikalisch kleinsten Vornamen zurückgeben. Nutzen Sie hier ebenfalls den Iterator. Der Iterator sollte als innere Klasse realisiert werden. Überlegen Sie, welchen Typ einer inneren Klasse Sie am besten verwenden.

2. Aufgabe
   Implementieren Sie zwei Klassen `NumberCruncherAnonym` und `NumberCruncherTopLevel`. Beide Klassen besitzen ein `float`-Array, welches über den Konstruktor übergeben wird. Außerdem müssen beide Klassen folgende Methoden besitzen:

   - `public void crunch(String[] operations)`: Diese Methode nimmt ein `String`-Array entgegen, welches Operationen angibt, die der Reihe nach auf das `float`-Array der Klasse angewendet werden müssen. Die Operationen sind unter c beschrieben.
   - `public float[] getNumbers()`: Diese Methode gibt das `float`-Array der Klasse zurück.

   Im Detail ist folgendes zu implementieren:

(a) Implementieren Sie die Klasse `NumberCruncherAnonym` mithilfe von anonymen Klassen, d.h. die einzelnen Operationen, die in Teil (c) beschrieben sind, sollen in Form von anonymen Klassen realisiert werden.

(b) Implementieren Sie die Klasse `NumberCruncherTopLevel` so, dass die einzelnen Operationen (siehe Teil (c)) in Form von Top-Level-Klassen realisiert werden. Vergleichen Sie beide Lösungen und identifizieren Sie die wesentlichen Unterschiede hinsichtlich der Struktur.

Jede der einzelnen Top-Level-Klassen muss das Interface `CrunchOperation` mit der Methode `void crunch(float values[])` implementieren. Koordiniert werden die Operationen von der Klasse `NumberCruncherTopLevel`.

(c) Folgende Operationen sollen möglich sein:

- **sum**: Summiert die Elemente des Arrays paarweise von links nach rechts auf und speichert den neuen Wert in dem jeweils rechten Datenfeld. D.h.: a[1] = a[0] + a[1]; a[2] = a[1] + a[2] usw.
- **swirl**: Führt n zufällige Vertauschungen der Datenfelder durch; n ist durch die Länge des float-Arrays gegeben.
- **divide**: Teilt die n/2 größten Werte im Array durch die n/2 Kleinsten und speichert den neuen Wert im Datenfeld des jeweils größeren Wertes. D.h. der größte Wert wird durch den Kleinsten geteilt. Der Zweitgrößte durch den Zweitkleinsten usw.
- **subtract**: Analog zu sum nur mit Subtraktion.
- **average**: Bestimmt den Durchschnitt aller Werte im Array und speichert den Durchschnittswert im Datenfeld mit dem größten Wert.

(d) Implementieren Sie einen Testdialog. Dieser muss die Möglichkeit bieten das `float`-Array, welches den Klassen über den Konstruktor übergeben wird, sowohl mit einzugebenden (manuell) als auch zufälligen Zahlen (automatisch) zu befüllen.