ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Science

Programmierung 2 - Sommersemester 2020

Prof. Dr. Peter Birkner

Übungsblatt Nr. 16
Abgabe KW 21

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

   ```
   interface PersonIterator extends java.util.Iterator<Person> { }
   ```

   Erweitern Sie die Klasse `PersonQueue` um eine Methode `String toString()`. 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. Ist die Warteschlange leer, dann soll ein leerer String zurückgegeben werden. Nutzen Sie hier ebenfalls den Iterator.

   Der Iterator soll als innere Klasse realisiert werden. Überlegen Sie, welchen Typ einer inneren Klasse Sie am besten verwenden.

2. Aufgabe
   Implementieren Sie die zwei Klassen `NumberCruncherAnonym` und `NumberCruncherTopLevel`. Im Detail:

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

   (b) Implementieren Sie eine zweite Variante, bei der die einzelnen Operationen (siehe Teil (d)) 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) Für die beiden Klassen `NumberCruncherTopLevel` und `NumberCruncherAnonym` gelten:
   Sie erhalten beide im Konstruktor ein float-Array mit den zu verarbeitenden Zahlen und bieten die Methode `public void crunch(String[] operations)` an. Das übergebene String-Array beinhaltet Operationen, welche der Reihe nachauf das float-Array der Klasse angewendet werden. Außerdem enthalten sie eine Methode `public float[] getNumbers()`, die das float-Array zurückgibt.

   (d) 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 Substruktion.
- **average**: Bestimmt den Durchschnitt aller Werte im Array und speichert den Durchschnittswert im Datenfeld mit dem größten Wert.

(e) Schreiben Sie einen Testdialog, der die Methoden mit zufällig erzeugten Zahlen aufruft. Dieser Testdialog ist verpflichtend. Optional können Sie zusätzlich JUnit-Tests erstellen.