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. 20
Abgabe KW 26

Allgemeiner Hinweis: Implementieren Sie für alle Ihre Lösungen einen Dialog-Test, auch wenn dies bei einzelnen Aufgaben nicht explizit in der Aufgabenstellung genannt ist. Der Test sollte so gestaltet sein, dass die Funktionsweise Ihres Programms bei der Abnahme gut nachvollzogen werden kann.

1. Aufgabe
   Implementieren Sie ein Programm mit einer Producer- und einer Consumer-Klasse. Der Producer erzeugt zufällige Integer, welche in einer geeigneten Collection gespeichert werden. Der Consumer entnimmt die Integer aus der Collection und berechnet die Quersumme. In einer Schleife sollen Producer und Consumer in einer zufälligen Reihenfolge aufgerufen werden. Etwa wie folgt:

```
Random ran = new Random();
for(int i = 0; i<10000; i++)
{
  if(ran.nextInt(2) > 0 )
    // Erzeugen eines neuen Integers durch den Producer und speichern
    // in einer Collection
  else
    // Entnehmen eines Integeres aus der Collection und Berechnung der
    // Quersumme durch den Consumer
}
```

   Beachten Sie bei der Implementierung folgende Aspekte:

   (a) Die Entnahme der Integer aus der Collection soll entweder nach FIFO-Ordnung oder entsprechend der natürlichen Sortierung der Integer erfolgen. Welche der beiden Varianten genutzt wird, soll beim Programmstart als Parameter übergeben werden. Überlegen Sie, welche Collection für die jeweilige Variante am besten geeignet ist.

   (b) Die Producer-Klasse besitzt eine Methode `produce` welche einen zufälligen Integer zwischen 0 und 1000 erzeugt und zurückgibt.

   (c) Die Consumer-Klasse besitzt folgende Methoden:

   - `consume`: Diese Methode nimmt einen Integer entgegen und berechnet die Quersumme. Für jede Berechnung soll außerdem der Zeitstempel der Berechnung gespeichert werden. Nutzen Sie die Methode `System.currentTimeMillis()`, um den Zeitstempel zu erhalten.
     Die Zeitstempel müssen so gespeichert werden, dass für eine gegebene Quersumme alle

zugehörigen Zeitstempel effizient zugegriffen werden können. Zugehörige Zeitstempel heißt: Die Zeitstempel aller Berechnungen, die zu der gegebenen Quersumme geführt haben. Es soll außerdem möglich sein effizient zu zählen, wie häufig eine gegebene Quersumme berechnet wurde. Überlegen Sie welche Datenstruktur dies am besten unterstützt. Hinweis: Sie müssen eine Map und eine Collection in geeigneter Weise miteinander kombinieren.

- `numberOfDifferentResults`: gibt an, wie viele unterschiedliche Quersummen berechnet wurden
- `numberOfOccurrences`: gibt für einen gegebenen Integer an, wie häufig dieser als Ergebnis einer Berechnung vorkam
- `getCrossTotalsAscending`: gibt eine Collection zurück, welche die berechneten Quersummen in aufsteigender Reihenfolge enthält
- `getCrossTotalsDescending`: gibt eine Collection zurück, welche die berechneten Quersummen in absteigender Reihenfolge enthält.
- `getTimestampsForResult`: nimmt einen Integer entgegen und gibt eine Collection zurück, welche alle zugehörigen Zeitstempel enthält. D.h. die Zeitstempel der Berechnungen, die zu dem gegebenen Ergebnis geführt haben.

Implementieren Sie eine Testklasse, um Ihre Implementierung zu testen.

Hinweis: Ein wichtiger Aspekt der Bewertung ihrer Lösung ist, ob Sie geeignete Datenstrukturen gewählt und diese in geeigneter Weise genutzt haben. Soweit es möglich ist, sollen Sie die Funktionalität der Java-Collections nutzen, um die Anforderungen umzusetzen; z.B., um die Zeitstempel in aufsteigender Reihenfolge zu erhalten.

2. Aufgabe
Ändern Sie die `Lager`-Klasse aus Übungsblatt 18 so, dass Sie zur Speicherung der Artikel eine `java.util.Map<K,V>` anstelle eines Arrays verwenden. Wählen Sie Ihre Datenstruktur so, dass Sie anhand der Artikelnummer in konstanter Zeit auf die Artikel zugreifen können. Außerdem sollte es möglich sein, in der Reihenfolge, in der die Artikel zum Lager hinzugefügt wurden, über die Artikel zu iterieren.

   Aufgrund der Umstellung auf eine Map werden einige Methoden, wie z.B. `erweitere_lager()`, nicht mehr benötigt, diese können Sie entfernen.

3. Aufgabe **(Zusatzaufgabe)**
Implementieren Sie einen MinHeap basierend auf einem Array fester Größe. In der Vorlesung Informatik 1 wurde dargestellt, wie dies sehr einfach zu realisieren ist. Beachten Sie dabei folgendes:

   (a) Ihr MinHeap sollte das Interface `java.util.Queue<E>` implementieren. Sie müssen jedoch nur folgende Interface-Methoden implementieren, bei allen anderen können Sie eine `java.lang.UnsupportedOperationException` werfen:
      i `public boolean offer(E e)`
      ii `public E poll()`
      iii `public E peek()`

   (b) Verwenden Sie nicht die abstrakte Klasse `java.util.AbstractQueue<E>`

   (c) Implementieren Sie den MinHeap als generische Klasse, welcher Objekte speichern kann, die das Interface `java.lang.Comparable` implementieren.

   (d) Implementieren Sie eine Testklasse, um die Methoden Ihres MinHeaps zu testen.