„Meine neuen Lieblings-Übungsaufgaben zu Funktionalem Programmieren“

(Zitat von ~~Vane~~… eines Studenten des WWI24B3-Kurses)

Aufgabe 4

Verwenden Sie für alle Aufgaben eine Klasse und wenn möglich Stream-Operationen:

a. Schreiben Sie eine passende Klassenmethode mit Namen **readTXT**, um aus einer Datei jede einzelne Zeile einzulesen, diese in einem String zu speichern und diesen String einer vorher neu erstellten **ArrayList<String>** hinzuzufügen.
Geben Sie diese fertige **ArrayList<String>** dann zurück (Rückgabewert).
Die Methode soll als formalen Parameter eine Variable vom Typ **File** haben und den **BufferedReader** zum Auslesen der Datei benutzen.
(Falls sie nicht schnell genug weiterkommen, nutzen Sie die auskommentierte Anweisung in der beigefügten Hilfsklasse)

b. Erstellen Sie in einer Klasse mit main-Methode mit Hilfe von a) eine **ArrayList<String>** aus der Textdatei „Personensatz.txt“
(Personensatz.txt liegt in OneDrive).
(Falls sie nicht schnell genug weiterkommen, nutzen Sie die auskommentierte Anweisung in der beigefügten Hilfsklasse)

c. Geben Sie die ersten 14 Daten der **ArrayList<String>** auf der Konsole aus.

d. Geben sie die ersten 5 Personen (Daten 2 bis 6) aus.

e. Schreiben Sie eine Klassenmethode
public static Person ausString (String s), welche einen String (so wie er in der ArrayList<String> vorhanden ist) in eine neue Variable des Typen Person (Person.java finden Sie im OneDrive-Ordner) „umwandelt“. Geben Sie diese Variable dann zurück.
Achten Sie auf die genauen Bezeichnungen des Personensatzes und der Variablen der Klasse Person (Anrede = anrede, usw…)
Verwenden Sie zum Aufteilen eines Strings die Methode
public String[] split(String trennzeichen) der Klasse String (siehe API).
(Falls sie nicht schnell genug weiterkommen, nutzen Sie die auskommentierte Anweisung in der beigefügten Hilfsklasse)

f. Konvertieren Sie alle Strings der ArrayList<String> als Typ Person (Person.java wie oben) und fügen diese in eine LinkedList<Person> daten2 ein – mit Hilfe der selbstgeschriebenen Klassenmethode aus e) und passenden Stream-Operationen.

(Ab hier die Liste daten2 verwenden)

g. Geben Sie alle Personen aus Rheinland-Pfalz aus.

h. Geben Sie die Anzahl der Personen, die älter als 50 Jahre alt sind, an. (public long count() der Klasse Stream hilft)

i. Sortieren Sie die Personen aufsteigend nach dem Alter, wobei nur die Zahl des Alters auf der Konsole ausgegeben werden soll, ergänzt mit dem Wort „Jahre".

j. Geben Sie alle Personen aus, die zwei aufeinanderfolgende „nn" in ihrem Nachnamen haben und achten Sie darauf, dass keine Person doppelt vorkommt.

k. Geben Sie den Durchschnitt der Zeichenlänge aller Nachnamen aus.

l. Ändern Sie das Bundesland aller Personen zu einem dreistelligen Ländercode (Ersten drei Buchstaben des Landes) und geben Sie die ersten 20 Personen auf dem Bildschirm aus.

m. Geben Sie aus, ob es Personen in der Liste gibt, welche zwei aufeinanderfolgende gleiche Vokale (aa/ee/...) in ihrem Vornamen haben.

n. Geben Sie aus, wie viel Prozent der Personen keine Anrede mit „Frau" oder „Herr" angegeben haben.

o. Schreiben sie "Volljährig" in die Konsole, wenn mehr als 2500 Personen bereits 18 Jahre alt sind, ansonsten schreiben Sie "Nicht volljährig".

p. Erstellen Sie einen passenden Stream, welcher die ganzen Zahlen von 1 bis 38 durchgeht und von den darin enthaltenen ungeraden Zahlen das Produkt berechnet. Das Ergebnis soll auf dem Bildschirm ausgegeben werden.

q. Erstellen Sie einen Stream aus unendlich vielen Integer-Zahlen. Dieser Stream soll beliebige Zufallszahlen zwischen (inklusive) 1 und 12 enthalten. Es sollen die ersten 20 Zahlen davon ausgewählt und überprüft werden, wie viele Zahlen ihres eigenen Geburtsmonats (in Zahlen ausgedrückt, wobei Januar die Zahl 1 ist und Dezember die Zahl 12) in diesen 20 Zahlen enthalten sind. Das Ergebnis soll in einer passenden Variablen gespeichert und dann auf dem Bildschirm ausgegeben werden.

r. Wir wollen mal über Performance reden. Bisher haben wir noch gar nicht feststellen können, ob Streams so viel besser sind. Dies wollen wir ändern 😊 Nehmen wir einen Algorithmus, der prüft, ob eine Zahl **number** eine Primzahl ist:

```
public static boolean isPrime(int number) {
  if (number == 2 || number == 3)
    return true;
  if (number % 2 == 0)
    return false;
  int sqrt = (int) Math.sqrt(number) + 1;
  for (int i = 3; i < sqrt; i += 2)
    if (number % i == 0)
      return false;

  return true;
}
```

Erstellen Sie sich also eine Klasse mit main-Methode (und obiger Klassenmethode) in welcher wir eine ArrayList<Integer> mit den Zahlen 1 bis 100.000.000 füllen. Wir verwenden dazu einen IntStream und die .collect-Methode wie folgt:

```
ArrayList<Integer> numbers = IntStream.rangeClosed(0, 100_000_000)
        .collect(ArrayList::new,ArrayList::add,ArrayList::addAll);
```

Wir wollen nun 3 Fälle testen und jeweils die Zeit messen. Die Zeit messen sie mit einem Date-Objekt vor und nach jedem Algorithmus und dem Befehl **.getTime** des Date-Objektes oder mit **System.currentTimeMillis()**.

Fall 1: for-Schleife oder forEach-Schleife, in der jedes Element getestet wird
Fall 2: Ein stream() angelegt an **numbers** mit passendem Filter und der Abschlussoperation **.count()**
Fall 3: Ein parallelStream() angelegt an **numbers** mit passendem Filter und der Abschlussoperation **.count()**

s. Was möchten Sie in der letzten-Vorlesung erleben? (frei denken)
Klausur / Poker / Tennis / Tanzvorstellung Ihrer Kurssprecherin / Jägi / etc…?

t. Implementieren Sie eine Methode **static generateFibonacci(int n)**, die eine **List<BigInteger>** zurückgibt. Diese Methode soll die ersten n Fibonacci-Zahlen generieren und in einer ArrayList<BigInteger> speichern.

   Die Fibonacci-Sequenz startet mit zwei Zahlen 1. Jede nachfolgende Zahl ist die Summe der beiden vorherigen Zahlen. Zum Beispiel: 1, 1, 2, 3, 5, 8, 13, 21, ...

   Verwenden Sie **BigInteger.ONE** für die Initialisierung der ersten beiden Zahlen.

   Fügen Sie in einer Schleife jede neue Zahl durch Addieren der letzten zwei Zahlen der Liste hinzu, indem Sie die Methoden **add()** und **get()** der BigInteger-Klasse verwenden.

   Verwenden Sie diese Methode um eine BigInteger-Liste zu erstellen mit einem Wert > 1000 und berechnen Sie mit der stream()-Methode die Summe der generierten Fibonacci-Zahlen zu berechnen.

   Drucken Sie die berechnete Summe auf der Konsole aus.