1. Java 8 - Streaming API

(a) Erzeuge einen `IntStream` aus den Zahlen 1 bis 20. Filtere so, dass die ungeraden Zahlen übrigbleiben und bilde diese mit einer geeigneten Operation auf ihre Quadratzahlen ab. Gib diese Quadratzahlen auf der Konsole aus.

(b) Berechne mit Hilfe des Streaming-API die folgende Summe:

$$\sum_{i=1}^{100} \frac{1}{(i+1)(i+2)}$$

(c) Erzeuge mit `generate(...)` einen unendlichen Stream aus möglichen Lottozahlen und generiere daraus einen sortierten Lottotipp.

(d) Lege ein ungeordnetes Integer-Array an, in dem gerade und ungerade Zahlen vorkommen. Zahlen dürfen (sollen) doppelt auftreten. Erzeugen daraus einen `IntStream`, der am Ende die Quadrate der ungeraden Zahlen genau einmal sortiert ausgibt.

(e) Verwende einen `LongStream` mit Reduktion, um 20! zu berechnen.

(f) Löse mit Hilfe des Streaming-API die folgende Aufgabe: Stelle fest, wie viele Ziffern '1' in jener Zahl enthalten sind, die man erhält, wenn man die Zahlen 1 bis 1000 verkettet: 123456789101112...9989991000
`Hinweis:` Man kann z.B. mit Hilfe von Reduktion einen String erzeugen, der die Zahl enthält und in diesem String dann die Einser zählen.

(g) Welche Zahlenfolge gibt die nachstehende Codezeile aus? Überlege zuerst auf ohne Rechner und überprüfe dann dein Ergebnis.

```
LongStream.iterate(1, i -> i * (i + 1))
          .limit(6)
          .forEach(System.out::println);
```

(h) Ein Supplier zur Generierung der Zahlenfolge der $n!$, $n = 1, 2, 3, \ldots$ kann z.B. wie folgt implementiert werden:

```
class FaktSupplier implements Supplier<BigInteger> {
  private int i = 0;
  private BigInteger faktorielle = BigInteger.ONE;

  @Override
  public BigInteger get() {
    faktorielle = faktorielle.multiply(BigInteger.valueOf(++i));
    return faktorielle;
  }
};
```

Erzeuge mit Hilfe dieses Suppliers einen Stream der Faktoriellen und stelle fest, welche Fakultät erstmalig länger als 99 Zeichen ist.

(i) Stelle nun jenes n fest, für das $n! > 10^{10000}$ ist.
Hinweis: der Supplier aus der obigen Aufgabe muss angepasst werden.

(j) Löse Euler-Aufgabe 25. Erstelle dazu einen Supplier für die Folge der Fibonaccizahlen.

Lösungen (zur Kontrolle):

```
Aufgabe 1
1 9 25 49 81 121 169 225 289 361

Aufgabe 2
0.49019607843137253

Aufgabe 3
5 12 22 33 34 40

Aufgabe 4
Array: [8, 6, 1, 8, 3, 9, 7, 3, 5, 13, 8, 1, 4, 5]
1 9 25 49 81 169

Aufgabe 5
2432902008176640000

Aufgabe 6
301

Aufgabe 7
1
2
6
42
1806
3263442

Aufgabe 8
119785716699698917960727837216890987364589381425464258575555
362864628009582789845319680000000000000000000

Aufgabe 9
3249
```

2. Java 8 - Streaming API

Gegeben sind eine Geschäftsklasse `data.Schueler` und eine Utilityklasse `data.SchuelerUtils` sowie eine CSV-Datei `schueler12.csv`. Löse unter Verwendung dieser Angaben die folgenden Aufgaben mit dem Java 8 Streaming-API:

(a) Bestimme die Anzahl der weiblichen Schüler der Testklasse.

(b) Die Vornamen der Schülerinnen in der Testklasse sind durch Beistriche getrennt in einem String zu verketten.

(c) Gib die Stringdarstellungen aller in der CSV-Datei vorkommenden Schülerinnen nach Klasse und Katalognummer aufsteigend sortiert zeilenweise aus.

(d) Man weiß, dass der Abeilungssprecher in eine 4.Klasse geht, männlich ist und mit Vornamen Lukas heißt. Durchsuche die CSV-Datei nach Schülern mit diesen Eigenschaften und gib die Stringdarstellungen der gefundenen Schüler aus.

(e) Stelle fest, aus wie vielen Zeichen der Längste Name (Zuname + Vorname) in der CSV-Datei besteht.

(f) Stelle fest, ob es in der CSV-Datei wenigstens eine Schülerin gibt, deren Vorname ''Julia'' enthält. Wenn ja, so ist die Stringdarstellung des ersten Fundes auszugeben, sonst lautet die Aufgabe ''keine Julia''.

(g) Es sind alle verschiedenen Klassenbezeichnungen aufsteigend sortiert durch Kommas getrennt in einem String zu verketten und dieser String ist auszugeben.

(h) Es ist eine Map zu erzeugen, in der die Schlüssel die verschiedenen Vornamen und die Werte die zugehörigen Schülerzahlen sind.
Hinweis: im Skriptum gibt es ein ähnliches Musterbeispiel.

Lösungen (zur Kontrolle):

```
Aufgabe 1
Anzahl der Schülerinnen: 3

Aufgabe 2
Sarah, Angela, Carina

Aufgabe 3
1AHIF/10 Hickelsberger-Fueller Sonja W
1AHIF/30 Traxler Tanja W
1AHIF/31 Woegerer Lisa-Marie W
1BHIF/05 Floh Elisabeth W
1BHIF/09 Hollerer Nadine W
1CHIF/10 Holasek Cornelia W
1CHIF/31 Wagner Nicole W
2BHIF/05 Fassl Nina Maria W
2BHIF/07 Hackl Desiree Iris W
2CHIF/12 Pfeiffer Patricia W
2CHIF/18 Stehling Lisa Maria W
2CHIF/20 Winter Sabine Elvira W
3BHIF/03 Ecker Andrea Katharina W
3BHIF/12 Schmatz Julia Andrea W
4BHIF/19 Wagner Nicole W
4CHIF/20 Zimmer Barbara W
5BHIF/02 Gruber Sarah W
5BHIF/09 Purker Angela W
5BHIF/11 Ringelhahn Carina-Anna W

Aufgabe 4
4CHIF/05 Kraushofer Lukas M
4BHIF/19 Schindlegger Lukas M

Aufgabe 5
40

Aufgabe 6
3BHIF/12 Schmatz Julia Andrea W

Aufgabe 7
1AHIF, 1BHIF, 1CHIF, 2AHIF, 2BHIF, 2CHIF, 3AHIF, 3BHIF, 4BHIF, 4CHIF,
5AHIF, 5BHIF


Aufagbe 8
{Albin=1, Alen=1, Alex Michael=1, Alexander=3, Alexander Florian=1, ...}
```

3. Serialisierung

In der CSV-Datei `GPS-Log.csv` stehen GPS-Daten zeilenweise wie folgt zur Verfügung: Uhrzeit, geographische Länge, geographischen Breite und Seehöhe. Trennzeichen ist `';'`.

Beispiel:

```
08:20:00;15.64435;48.20155;305.6
08:20:05;15.66479;48.29093;308.6
08:20:10;15.73713;48.36192;311.2
08:20:15;15.74192;48.41317;311.1
...
```

Erstelle eine serializierbare Klasse `TrackPoint`, in der die oben beschriebenen Informationen gekapselt sind. Erstelle weiter eine Klasse `Tools` mit den folgenden statischen Methoden:

a) Die Methode
```
public static List<TrackPoint> readCSV(String filename)
                               throws IOException
```
soll alle Zeilen aus der Datei `GPS-Log.csv` lesen, eine `ArrayList` mit entsprechenden `TrackPoint`-Objekten erzeugen und als Returnwert liefern. Alle fehlerhaften Zeilen in der CSV-Datei sind herauszufiltern und während des Lesevorganges auf der Konsole zu protokollieren. Am Ende des Lesevorganges ist die Anzahl der fehlerhaften Zeilen auszugeben.

b) Die Methode
```
public static void serialize(List<TrackPoint> tp, String filename)
                               throws IOException
```
serialisiert die erfolgreich eingelesenen Daten in eine Datei `filename`.

c) Die Methode
```
public static void double maxElevation(String filename)
                               throws IOException
```
liest die serialisierten Daten aus der Datei und liefert die insgesamt größte Seehöhe.

d) Schreibe ein Testprogramm in einer eigenen Klasse `TrackPointApp`.