1. Leibnitzreihe

Die näherungsweise Berechnung von π kann nach Leibnitz wie folgt erfolgen:

$$\pi = 4 \cdot \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = 4 \cdot \left(1 - \frac{1}{3} + \frac{1}{5} - \ldots\right)$$

- Schreibe ein Programm, das die Summe der ersten 100000 Elemente also bis inklusive $n = 99999$ dieser Reihe berechnet.
- Die Berechnung erfolgt parallel in mehreren Threads, wobei jeder Thread eine entsprechende Teilsumme berechnet. Nach der Threaderzeugung sind das Intervall und die Anzahl der vom jeweiligen Thread durchgeführten Schleifendurchläufe anzugeben.
- Im Hauptthread ist mit Hilfe von `join()`-Aufrufen ein Synchronisationspunkt zu setzen und dann sind alle Teilsummen zusammenzuführen.
- Mit Hilfe einer Instanzvariablen vom Typ `boolean` ist in den Berechnungsthreads sicherzustellen, dass der entsprechende Getter erst dann ein Ergebnis liefert, wenn die Berechnung abgeschlossen ist. Bei einem vorzeitigem Aufruf wirft die Methode eine `IllegalStateException`.

Beispielausgabe für 7 Berechnungsthreads:

```
Thread-0: [    0, 14285] => 14286
Thread-1: [14286, 28571] => 14286
Thread-2: [28572, 42857] => 14286
Thread-3: [42858, 57143] => 14286
Thread-4: [57144, 71429] => 14286
Thread-5: [71430, 85714] => 14285
Thread-6: [85715, 99999] => 14285
pi = 3.1415826535897837
```

2. Paralleler Primzahlentest

Schreibe ein Programm, das prüft, welche positive Ganzzahlen in einem gegebenen Intervall Primzahlen sind. Entdeckte Primzahlen werden in eine konkurrierend genutzte Collection geschrieben. Der Primzahlentest ist für jede einzelne Zahl in einem eigenen Thread durchzuführen. Gehe zur Lösung dieser Aufgabe wie folgt vor:

- Implementiere eine eigene `Runnable`-Klasse `PrimeTest`, die in ihrer Methode `run()` den Primzahltest für genau positive Ganzzahl durchführt. Die zu prüfende Zahl ist im Konstruktor zu übergeben.
- Teste in einer Schleife, ob die Division n/k für $2 \leq k \leq \sqrt{n}$ nie den Rest Null liefert. Um die unterschiedliche Ausführungsdauer der jeweiligen Threads hervorzuheben, soll nur ein Schleifendurchlauf pro Sekunde ausgeführt werden. Versetze dazu die einzelnen Threads nach jeder Probedivision für `delay ms` (ein guter Wert für `delay` ist z.B. 1000) in den Zustand sleeping.

- Implementiere und starte zusätzlich einen Daemonthread, der ca. jede halbe Sekunde die alle bisher gefundenen Primzahlen aufsteigend sortiert sowie die Anzahl der (noch) aktiven Berechnungsthreads ausgibt. Die Priorität des Hintegrundthreads soll so niedrig wie möglich sein.

Anleitung:

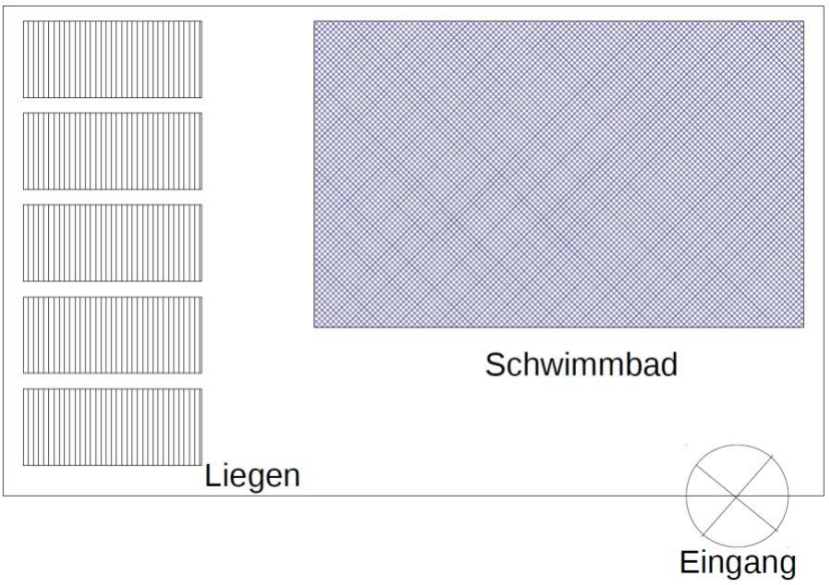
Die Anzahl der aktiven Berechnungsthreads kann man mit Hilfe einer statischen Variablen der Klasse `PrimeTest` feststellen. Wenn man sauber arbeitet, ist jeder Zugriff auf diese statische Variable zu synchronisieren sowie auf die Collection zu synchronisieren.

Beispielausgabe für das Intervall `[1, 50]`:

```
Active Checkers: 25
[2, 3, 5, 7]
Active Checkers: 7
[2, 3, 5, 7, 11, 13, 17, 19, 23]
Active Checkers: 7[2, 3, 5, 7, 11, 13, 17, 19, 23]
-------------------------------------------------------
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

3. Schwimmbadsimulation

Verwende einen Monitor, um folgende Schwimmbadsimulation zu implementieren (Badegäste sollen dabei als Java Threads realisiert werden):



- Das kleine Schwimmbad besitzt `maxLiegen`, die von den Badegästen genutzt werden können. Jeder Badegast belegt dabei genau eine Liege.
- Die Badegäste können das Schwimmbad über ein Drehkreuz betreten und verlassen. Dabei kann zu einem bestimmten Zeitpunkt immer nur eine Person durch das Drehkreuz gehen.
- Es dürfen sich nicht mehr Schwimmgäste im Bad befinden (und das Drehkreuz benutzen) als Liegen vorhanden sind.

Implementiere die Simulation für 5 Liegen und 50 Badegäste. Die Badegäste sollen eine zufällige Zeitspanne im Intervall `[500, 5500]ms` warten, bis sie das Schwimmbad betreten und dann eine zufällige Zeitspanne im Intervall `[2000, 7000]ms` im Schwimmbad verweilen. Die Tickets sollen in der Reihenfolge ihres Erscheinens von 1 bis 50 durchnummeriert werden.

- Schreibe für dieses Szenario ein Java-Programm, welches den 50 Badegäste den synchronisierten Einlass gewährt, so dass obigen Bedingungen nicht verletzt werden!
- Verwende zur Lösung die folgenden Klassen:
  - `Badegast extends Thread`
    Will nach zufälliger Zeitspanne Schwimmbad betreten und nach einer zufälligen Verweilzeit wieder verlassen
  - `Simulation`
    Erzeugt in ihrer `main()`-Methode das Schwimmbad, die Badegäste und schickt sie ins Schwimmbad. Wenn der letzte Badegast das Schwimmbad verlassen hat ist eine Schlussmeldung auszugeben.
  - `Schwimmbad`
    Hat eine Instanzvariable für die Anzahl der Liegen, verwaltet die Ticketnummern und stellt Methoden zum Betreten und Verlassen des Schwimmbades bereit.

Mögliche Ausgabe:

```
Badegast_4 betritt Schwimmbad mit Ticket 1 ---> (1 anwesend)
Badegast_39 betritt Schwimmbad mit Ticket 2 ---> (2 anwesend)
Badegast_38 betritt Schwimmbad mit Ticket 3 ---> (3 anwesend)
Badegast_25 betritt Schwimmbad mit Ticket 4 ---> (4 anwesend)
Badegast_44 betritt Schwimmbad mit Ticket 5 ---> (5 anwesend)
Badegast_4 verlaesst Schwimmbad  <--- (4 anwesend)
...
Badegast_28 verlaesst Schwimmbad  <--- (4 anwesend)
Badegast_8 betritt Schwimmbad mit Ticket 50 ---> (5 anwesend)
Badegast_20 verlaesst Schwimmbad  <--- (4 anwesend)
Badegast_36 verlaesst Schwimmbad  <--- (3 anwesend)
Badegast_9 verlaesst Schwimmbad  <--- (2 anwesend)
Badegast_7 verlaesst Schwimmbad  <--- (1 anwesend)
Badegast_8 verlaesst Schwimmbad  <--- (0 anwesend)
Alle Gäste haben das Bad verlassen
```

4. Algorithmik - minimaler Spannbaum

Ein minimaler Spannbaum in einem Grafen mit n Knoten Eulerproblem 107 - Minimales Netzwerk kann nach dem Kurskal-Algorithmus wie folgt berechnet werden:

(a) Man sortiert die Kanten aufsteigend nach ihrer Gewichtung.

(b) Man speichert jeden Knoten in einem eigenen Gebiet, zu Beginn hat man also n Gebiete mit je einem Knoten

(c) Wiederhole n - 1 Mal:

- Bestimme die nächste nicht verbrauchte Kante mit der kleinsten Gewichtung
- Liegen die beiden Knoten dieser Kante in verschiedenen Gebieten, so vereinige die beiden Gebiete und speichere die Kante, andernfalls ignoriere den Schritt

(d) Nun hat man nur mehr ein Gebiet und n-1 Kanten. Dieser Graph repräsentiert den minimalen Spannbaum.

Löse mit diesem Algorithmus Euleraufgabe 107.