Name: _____ Vorname: _____

Matrikelnummer: _____ Testat:_____

Paradigmen der Programmierung
Praktikum 4

Hinweise für die Informationssuche:

1. Schauen Sie ins Skript und in die Vorlesungsfolien
2. Nutzen Sie die Java Dokumentation:
   http://docs.oracle.com/javase/8/docs/api/overview-summary.html

Aufgabe 1
Was ist der Unterschiede zwischen Threads, Runnables, Callables, Futures, und Executors? Erklären Sie die Unterschiede anhand einer selbst angefertigten Skizze.

Aufgabe 2
In den nächsten Aufgaben soll die nebenläufige Berechnung von Primzahlen mithilfe des Executor-Frameworks gelöst werden.

Gegeben ist die Klasse `FindPrimeNumbers`. Bei der Objekterzeugung wird dem Konstruktor ein Zahlenbereich übergeben. Die Methode `doCalculations()` findet alle Primzahlen innerhalb des Bereichs.

Die beiden Dateien (package beachten!) sind hier auffindbar:
http://www.kohls.de/wp-content/uploads/2014/11/concurrentcalc.zip

Die `main()` Methode teilt einen gegebenen Wertebereich auf und erzeugt mehrere `FindPrimeNumber`-Objekte für die aufeinanderfolgenden Segmente. Die Berechnung erfolgt dann mit `doCalculations()` für die einzelnen Objekte. Da es sich noch nicht um eine nebenläufige Lösung handelt, kann nach Beenden der `doCalculations()`-Methode einfach das Objekt als Ergebnis ausgegeben werden, denn `toString()` gibt alle gefundenen Primzahlen des Wertebereichs als einen String zurück.

Aufgabe 3
- Implementieren Sie das `Runnable` Interface für die Klasse `FindPrimeNumbers`
- Wandeln Sie `doCalculations()` in eine private Hilfsmethode, die in `run()` aufgerufen wird.
- Implementieren Sie eine neue Klasse, die das `Executor` Interface implementiert. Diese Implementierung soll zunächst als SingleThread-Lösung umgesetzt werden.

Hinweis: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executor.html

Implementieren Sie nun zusätzlich das Executor Interface als ThreadPerTask, d.h. für jede Aufgabe wird ein neuer Thread erzeugt.

Warum liefert die Zeile

```
System.out.println(pn);
```

nicht mehr wie vorher den gewünschten Effekt? Wie kann man dies lösen?

Aufgabe 4
Implementieren Sie nun das Executor Interface als Thread-Pool, d.h. Sie verteilen die Aufgaben auf eine bestimmte Anzahl Threads. Die Anzahl soll als Konstante in der Implementierung festgelegt werden, z.B.

```
final int NTHREADS = 20;
```

Tipps für die Threadpool-Umsetzung:

- Die Threads können gleich beim Erzeugen der Thread-Pool-Lösung angelegt werden.
- Verwenden Sie eine vorhandene `BlockingQueue` Implementierung, um neue Arbeitsaufträge in einer Warteschlange zu puffern.
- Jeder Thread läuft in einer Endlosschleife und holt sich den nächsten Arbeitsauftrag aus der Warteschlange

Hinweis: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html

Optionale Zusatzaufgaben
Ändern Sie die `main()`-Methode so, dass leicht konfigurierbar ist, welcher Executor verwendet wird. Messen Sie die Systemzeit mit `System.nanoTime()` am Anfang und am Ende der Berechnung für die verschiedenen Executor-Implementierungen. Was sind die Ergebnisse?

Die Schleife in `main()` teilt die Berechnung von Primzahlen in gleich große Abschnitte ab. Warum ist dies für die parallele Berechnung gerade von Primzahlen nicht die beste Aufteilung?

Erstellen Sie eine Klasse `PrimeResults`, in der Primzahlen gespeichert werden können. Intern kann hierzu eine beliebige Containerklasse verwendet werden. Verschiedene Threads können nun Primzahlen an PrimeResults melden.

- `PrimeResults` muss threadsicher implementiert sein
- Erstellen Sie eine Funktion `Iterator <Integer> getNPrimes(int n)`, die n Primzahlen als Iterator zurückgibt
- Wenn n größer ist als die Anzahl der bereits berechneten Primzahlen ist, dann soll `getNPrimes` solange blockieren bis entsprechend viele Primzahlen vorhanden sind.
- Verwenden Sie z.B. eine Semaphore, um zu gewährleisten, dass erst auf n Primzahlen zugegriffen wird sobald so viele auch vorhanden sind.

Hinweis: https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Semaphore.html