# Java SE 8 Programming


Robotron Datenbank-Software GmbH
Schulungszentrum
Heilbronner Straße 21
01189 Dresden

# Agenda

Java kann Aufgaben **effizient auf Kerne aufteilen**.

**Bitte wählen Sie eine Kategorie, um näheres zu erfahren!**

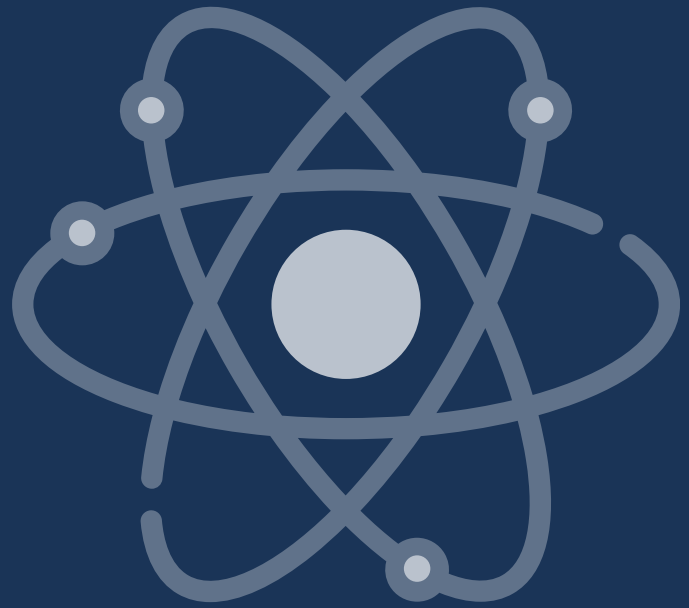


Mit Klick auf dieses Symbol gelangen Sie auf dieses Seite.

Mit diesen Symbolen navigieren Sie vor und zurück.

# KERNENERGIE?

## JA BITTE!




**Java** ist eine der am weitesten verbreiteten Programmiersprachen. Es bietet komfortable Möglichkeiten zur Parallelisierung des Programmablaufes.

Anwendbar sind diese Features für **parallelisierbare Aufgaben**. Ein Beispiel ist die Bearbeitung großer, aufteilbarer Datenmengen.

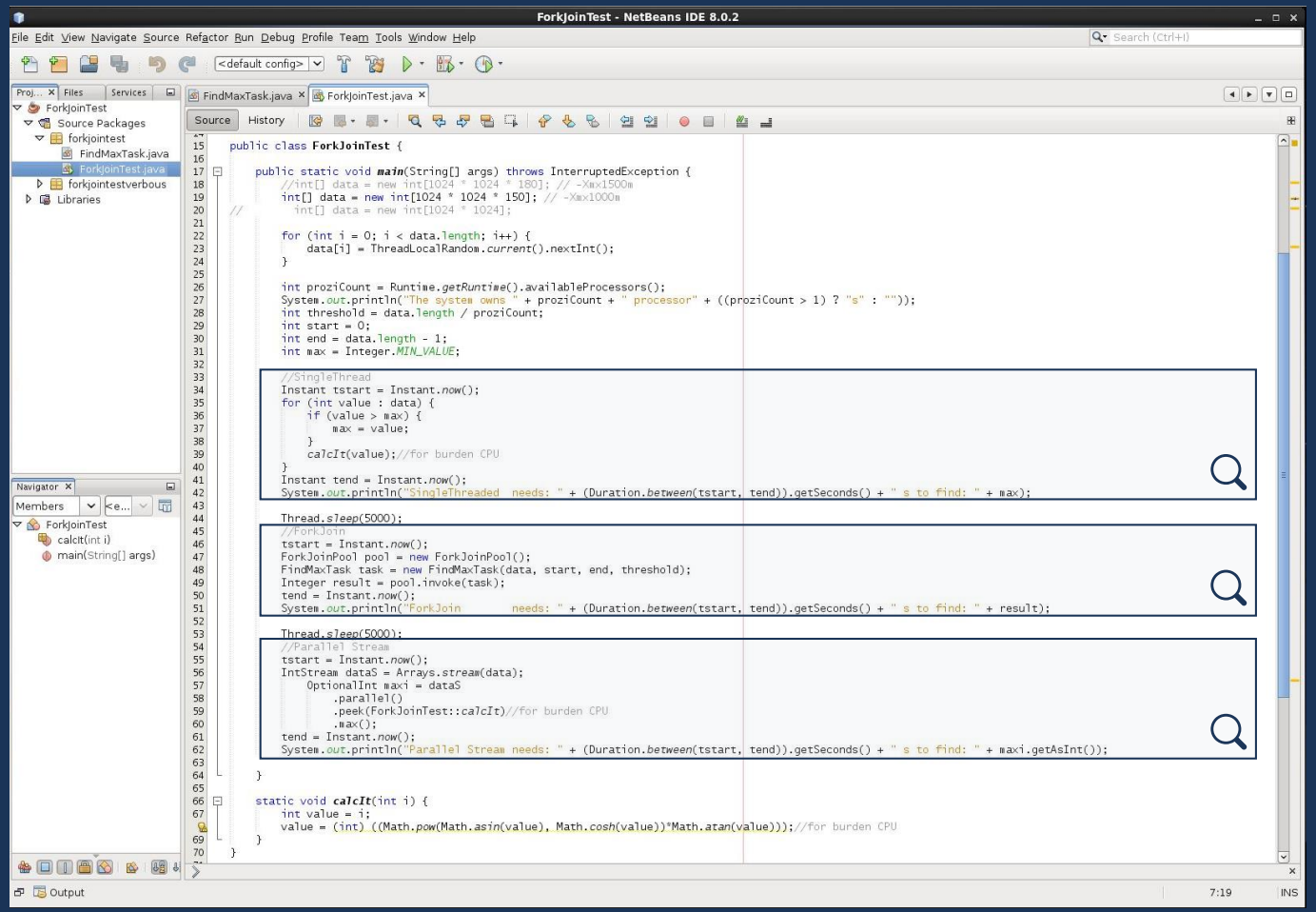
Da hierbei lediglich die Verteilung der Aufgabe auf die Ressource Kerne unterstützt wird, sollte die Performance der Aufgabe hauptsächlich von dieser Ressource abhängen. Eine Aufgabe mit geringer CPU-Last und z.B. hoher Lese-Schreiblast kann nicht profitieren, evtl. sogar langsamer werden.

# Aufgabenstellung

Die Suche eines Maximalwertes in einer größeren Datenmenge ist eine durch Aufteilung gut parallelisierbare Aufgabe.

Die Methode calcIt(int i) dient nur dazu, den Vorgang CPU-lastig zu machen.

**Der Klick auf die Lupen zeigt den Code vergrößert.**



# Aufgabenstellung

robotron schulung

… zuerst **Seriell (Single Threaded)**

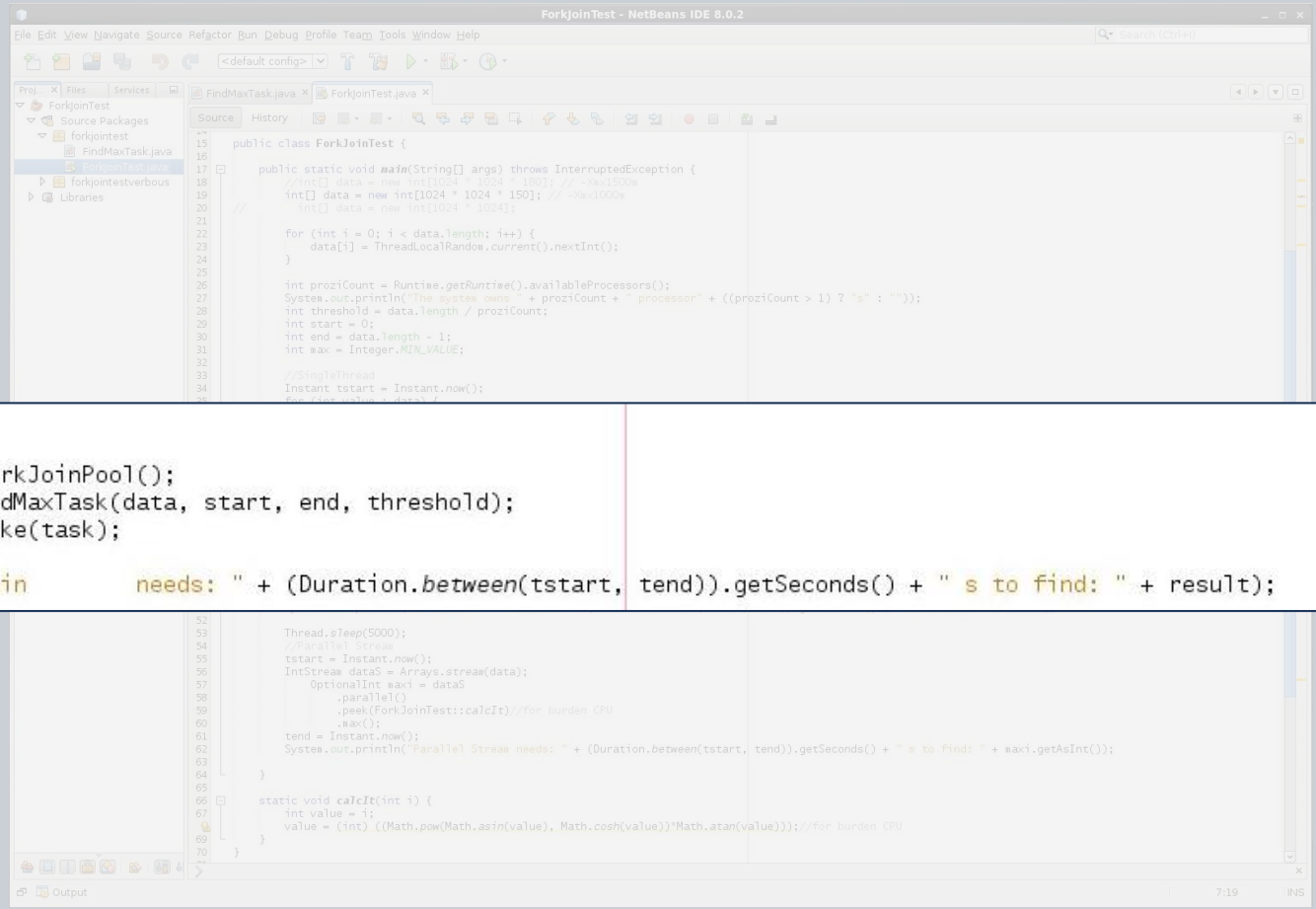
… nach den Code-abschnitten gibt es jeweils 5 Sekunden Erholung …

```
//SingleThread
Instant tstart = Instant.now();
for (int value : data) {
    if (value > max) {
        max = value;
    }
    calcIt(value);//for burden CPU
}
Instant tend = Instant.now();
System.out.println("SingleThreaded  needs: " + (Duration.between(tstart, tend)).getSeconds() + " s to find: " + max);

Thread.sleep(5000);
```

# Aufgabenstellung

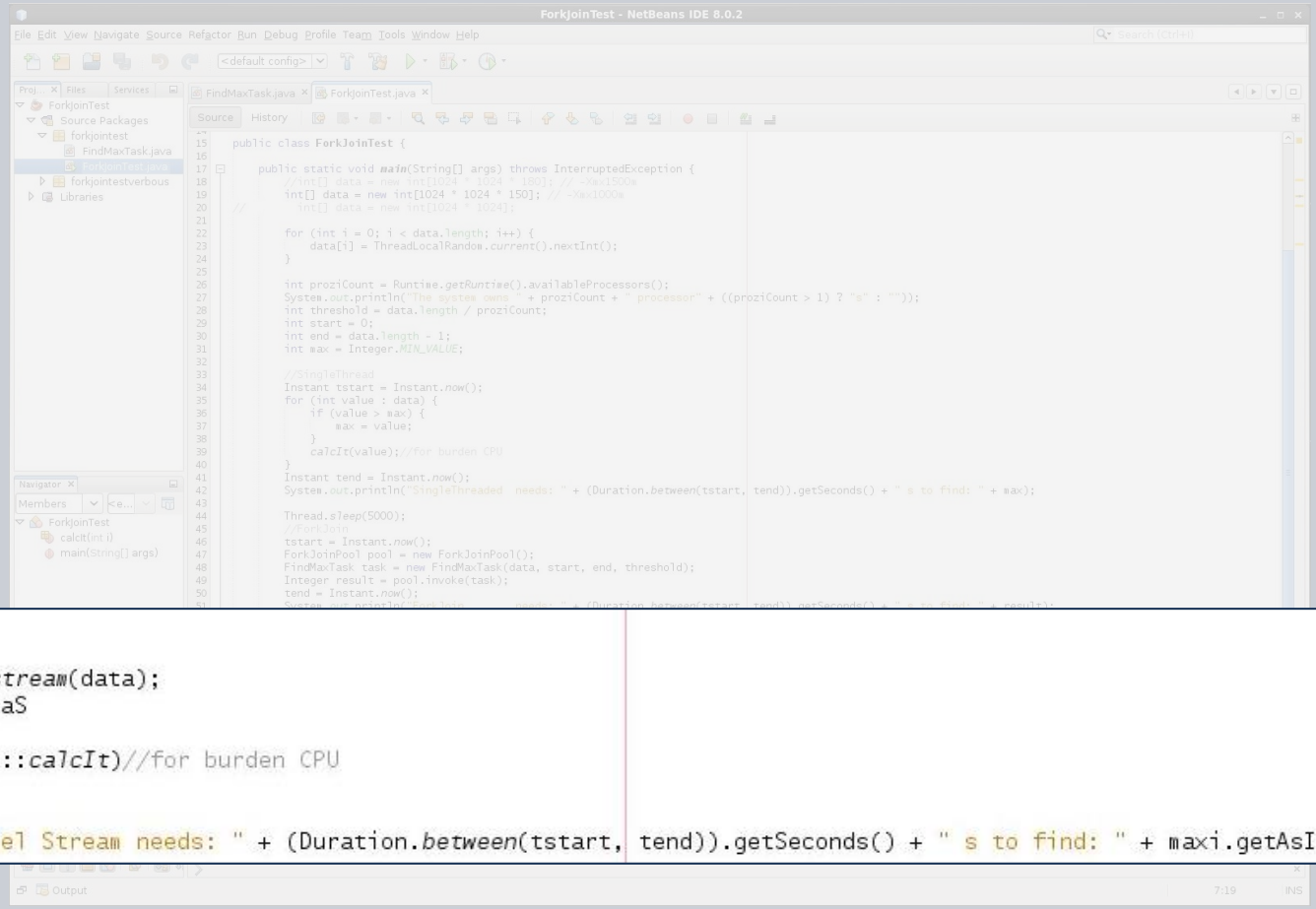
… jetzt parallel mittels **ForkJoin Framework** …




```
//ForkJoin
tstart = Instant.now();
ForkJoinPool pool = new ForkJoinPool();
FindMaxTask task = new FindMaxTask(data, start, end, threshold);
Integer result = pool.invoke(task);
tend = Instant.now();
System.out.println("ForkJoin        needs: " + (Duration.between(tstart, tend)).getSeconds() + " s to find: " + result);
```

# Aufgabenstellung

… und nun ebenfalls parallel mit der **Stream API** …




```
//Parallel Stream
tstart = Instant.now();
IntStream dataS = Arrays.stream(data);
    OptionalInt maxi = dataS
        .parallel()
        .peek(ForkJoinTest::calcIt)//for burden CPU
        .max();
tend = Instant.now();
System.out.println("Parallel Stream needs: " + (Duration.between(tstart, tend)).getSeconds() + " s to find: " + maxi.getAsInt());
```

# Ergebnis

Und wie sieht das der **Rechner**?

```
ForkJoinTest - NetBeans IDE 8.0.2

int start = 0;
int end = data.length - 1;
int max = Integer.MIN_VALUE;

1  //SingleThread
Instant tstart = Instant.now();
for (int value : data) {
    if (value > max) {
        max = value;
    }
    calcIt(value);//for burden CPU
}
Instant tend = Instant.now();
System.out.println("SingleThreaded  needs: " + (Duration.between(tstart, tend)).getSeconds() + " s to find: " + max);

Thread.sleep(5000);
2  //ForkJoin
tstart = Instant.now();
ForkJoinPool pool = new ForkJoinPool();
FindMaxTask task = new FindMaxTask(data, start, end, threshold);
Integer result = pool.invoke(task);
tend = Instant.now();
System.out.println("ForkJoin        needs: " + (Duration.between(tstart, tend)).getSeconds() + " s to find: " + result);

Thread.sleep(5000);
3  //Parallel Stream
tstart = Instant.now();
IntStream dataS = Arrays.stream(data);
    OptionalInt maxi = dataS
        .parallel()
        .peek(ForkJoinTest::calcIt)//for burden CPU
        .max();
tend = Instant.now();
System.out.println("Parallel Stream needs: " + (Duration.between(tstart, tend)).getSeconds() + " s to find: " + maxi.getAsInt());

static void calcIt(int i) {
    int value = i;
    value = (int) ((Math.pow(Math.asin(value), Math.cosh(value))*Math.atan(value)));//for burden CPU
}
```

```
run:
The system owns 4 processors
SingleThreaded  needs: 15 s to find: 2147483630
ForkJoin        needs: 6 s to find: 2147483630
Parallel Stream needs: 4 s to find: 2147483630
BUILD SUCCESSFUL (total time: 36 seconds)
```

System Monitor

Monitor Edit View Help

System | Processes | Resources | File Systems

CPU History — CPU1 2.0% | CPU2 1.0% | CPU3 2.0% | CPU4 0.0%

Memory and Swap History

Memory 1.0 GiB (27.9 %) of 3.7 GiB

Swap 0 bytes (0.0 %) of 4.0 GiB

Network History

Receiving 0 bytes/s — Total Received 0 bytes

Sending 0 bytes/s — Total Sent 0 bytes

# Besuchen Sie uns!


Wir bieten Ihnen viele weitere Kurse an und passend zum Thema folgende:

Java SE 8 Fundamentals

Java EE 7: Front-end Web Application Development

## Wir freuen uns auf Sie in unserem modern ausgestatteten Schulungszentrum!

Kontakt:

Michaela Eisold, Schulungsassistentin
Telefon: +49 351 25859-2660

E-Mail: schulung@robotron.de
Internet: www.robotron.de