Java: Fork-Join

Fork-Join-Pattern

```
LÖSE-PROBLEM (P)
     Wenn P klein genug
         Löse P sequenziell
     Sonst
         Teile P in kleinerer Teilprobleme P1 und P2
         Fork: LÖSE-PROBLEM (P1)
         Fork: LÖSE-PROBLEM (P2)
         Join
```

java.util.concurrent.RecursiveAction (ohne Rückgabewert)

RecursiveAction.java

```
class MyRecursiveAction extends RecursiveAction<INPUTDATA> {
    private INPUTDATA data;

    public MyRecursiveAction(INPUTDATA data) {
        this.data = data;
    }

    protected void compute() {
        if (data klein genug) {
            // sequenzielle Berechnung
        } else {
            // data aufteilen in data1 und data2

            // Beide Tasks parallel ausführen
            MyRecursiveAction task1 = new MyRecursiveAction(data1);
            MyRecursiveAction task2 = new MyRecursiveAction(data2);
            invokeAll(task1, task2);
        }
    }
}
```

Aufruf

```
MyRecursiveAction rootTask = new MyRecursiveAction(data);
rootTask.invoke();
```

java.util.concurrent.RecursiveTask (mit Rückgabewert)

MyRecursiveTask.java

```
class MyRecursiveTask extends RecursiveTask<INPUTDATA> {
    private INPUTDATA data;

    public MyRecursiveTask(INPUTDATA data) {
        this.data = data;
    }

    protected RETURNDATA compute() {
        if (data klein genug) {
            // sequenzielle Berechnung
        } else {
            // data aufteilen in data1 und data2

            // Beide Tasks parallel ausführen
            MyRecursiveTask task1 = new MyRecursiveTask(data1);
            MyRecursiveTask task2 = new MyRecursiveTask(data2);
            invokeAll(task1, task2);

            // Ergebnisse der beiden Tasks wieder zusammenführen
            RETURNDATA result = zusammenbauen(task1.join(), task2.join());
            return result;
        }
    }
}
```

Aufruf

```
MyRecursiveTask rootTask = new MyRecursiveTask(data);
rootTask.invoke();
RETURNDATA result = rootTask.join();
```