

Industrial strength Java/JEE Career Companion to open more doors


[Home](#)
[Java FAQs](#)
[600+ Java Q&As](#)
[Career](#)
[Tutorials](#)
[Member](#)
[Why?](#)
[Can u Debug?](#)
[Java 8 ready?](#)
[Top X](#)
[Productivity Tools](#)
[Judging Experience?](#)

[Home](#) › [Interview](#) › [Core Java Interview Q&A](#) › [Java 7](#) › Java 7 fork and join tutorial with a diagram and an example

# Java 7 fork and join tutorial with a diagram and an example

Posted on [November 25, 2014](#) by [Arulkumaran Kumaraswamipillai](#) — No

[Comments](#) ↓

0  
Like  
Share

Tweet

0  
G+1

Share

**Q.** How does `java.util.concurrent.RecursiveTask` work?

**A.** It uses a single `ForkJoin` pool to execute tasks in parallel for both computing sum of numbers recursively and *chunking* (or *breaking*) them into chunks of say 3 or less as per the following example.

Here is an example, where a an array of given numbers are summed in parallel by multiple threads. The batch size is 3. So, each thread processes 3 or less numbers asynchronously (i.e. **fork**) and the **join** to compute the final results.

**600+ Full Stack Java/JEE Interview Q&As ♥Free ♦FAQs**

[open all](#) | [close all](#)

[Ice Breaker Interview](#)

[Core Java Interview C](#)

[Java Overview \(4\)](#)

[Data types \(6\)](#)

[constructors-methc](#)

[Reserved Key Wor](#)

[Classes \(3\)](#)

[Objects \(8\)](#)

[OOP \(10\)](#)

[GC \(2\)](#)

[Generics \(5\)](#)

[FP \(8\)](#)

[IO \(7\)](#)

[Multithreading \(12\)](#)

[Algorithms \(5\)](#)

[Annotations \(2\)](#)

[Collection and Dat](#)

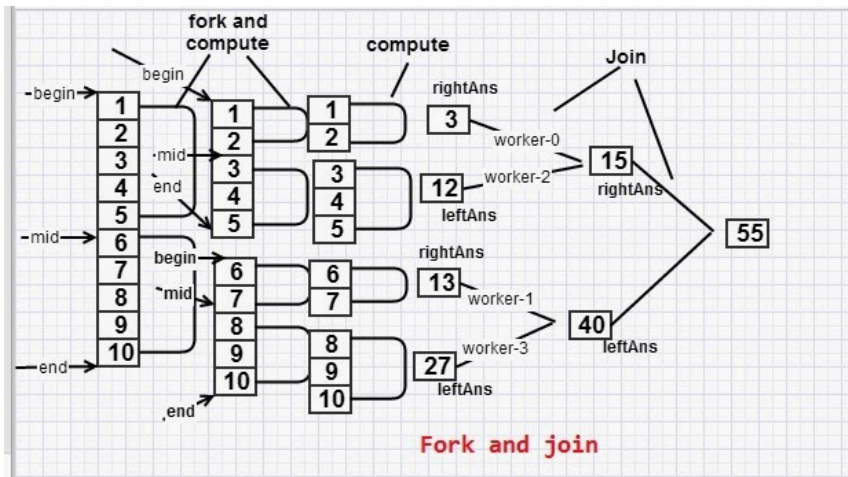
[Differences Betwee](#)

[Event Driven Progr](#)

[Exceptions \(2\)](#)

[Java 7 \(2\)](#)

Example. **numbers** = {1,2,3,4,5,6,7,8,9,10}, **sum** = 55;  
process them using the **fork and join** feature introduced in Java 7.



Here is the code

```

1 package com.fork.join;
2
3 import java.util.ArrayList;
4 import java.util.Arrays;
5 import java.util.List;
6 import java.util.concurrent.RecursiveTask;
7
8 class SumTask extends RecursiveTask<Integer> {
9
10     static final int CHUNK_SIZE = 3; // execution b
11
12     Integer[] numbers;
13     int begin;
14     int end;
15
16     SumTask(Integer[] numbers, int begin, int end) {
17         this.numbers = numbers;
18         this.begin = begin;
19         this.end = end;
20     }
21
22     @Override
23     protected Integer compute() {
24         //sums the given number
25         if (end - begin <= CHUNK_SIZE) {
26             int sum = 0;
27             List<Integer> processedNumbers = new Ar
28             for(int i=begin; i < end; ++i) {
29                 processedNumbers.add(numbers[i]); //just to t
30                 sum += numbers[i];
31             }
32
33             //tracking thread, numbers processed, and sum
34             System.out.println(Thread.currentThread().get
35                 Arrays.asList(processedNumbers) + ", sum
36             return sum;
37         }

```

[Java 7 fork and j](#)  
[Java 7: Top 8 ne](#)  
[Java 8 \(24\)](#)  
[JVM \(6\)](#)  
[Reactive Programn](#)  
[Swing & AWT \(2\)](#)  
[JEE Interview Q&A \(3](#)  
[Pressed for time? Jav](#)  
[SQL, XML, UML, JSC](#)  
[Hadoop & BigData Int](#)  
[Java Architecture Inte](#)  
[Scala Interview Q&As](#)  
[Spring, Hibernate, & I](#)  
[Testing & Profiling/Sa](#)  
[Other Interview Q&A 1](#)  
[Free Java Interview](#)

## 16 Technical Key Areas

open all | close all

[Best Practice \(6\)](#)  
[Coding \(26\)](#)  
[Concurrency \(6\)](#)  
[Design Concepts \(7\)](#)  
[Design Patterns \(11\)](#)  
[Exception Handling \(3](#)  
[Java Debugging \(21\)](#)  
[Judging Experience I](#)  
[Low Latency \(7\)](#)  
[Memory Managemen](#)  
[Performance \(13\)](#)  
[QoS \(8\)](#)  
[Scalability \(4\)](#)  
[SDLC \(6\)](#)  
[Security \(13\)](#)  
[Transaction Managen](#)

```

38
39 //create chunks, fork and join
40 else {
41     int mid = begin + (end - begin) / 2; //mid po
42     SumTask left = new SumTask(numbers, begin, m
43     SumTask right = new SumTask(numbers, mid, end
44     left.fork(); //asynchronously exec
45     int leftAns = right.compute();
46     int rightAns = left.join(); //returns the as
47     System.out.println("leftAns=" + leftAns + " +
48     return leftAns + rightAns;
49 }
50
51 }
52 }
53

```

Here is the test class with the main method

```

1 package com.fork.join;
2
3 import java.util.concurrent.ForkJoinPool;
4
5 public class SumTaskTest {
6
7     public static void main(String[] args) {
8         int numberOfCpuCores = Runtime.getRuntime().av
9         ForkJoinPool forkJoinPool = new ForkJoinPool(n
10
11         Integer[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8,
12         int sum = forkJoinPool.invoke(new SumTask(numb
13
14         System.out.println(sum);
15     }
16
17 }
18

```

### The output

```

ForkJoinPool-1-worker-1 proceesing [[6, 7]], sum = 13
ForkJoinPool-1-worker-3 proceesing [[8, 9, 10]], sum = 27
leftAns=27 + rightAns=13
ForkJoinPool-1-worker-2 proceesing [[3, 4, 5]], sum = 12
ForkJoinPool-1-worker-0 proceesing [[1, 2]], sum = 3
leftAns=12 + rightAns=3
leftAns=40 + rightAns=15
55

```

**Q.** Where to use fork/join as opposed to using the **ExecutorService** framework?

## 80+ step by step Java Tutorials

[open all](#) | [close all](#)

- [Setting up Tutorial \(6\)](#)
- [Tutorial - Diagnosis \(2\)](#)
- [Akka Tutorial \(9\)](#)
- [Core Java Tutorials \(2\)](#)
- [Hadoop & Spark Tuto](#)
- [JEE Tutorials \(19\)](#)
- [Scala Tutorials \(1\)](#)
- [Spring & Hibernate Ti](#)
- [Tools Tutorials \(19\)](#)
- [Other Tutorials \(45\)](#)

## 100+ Java pre-interview coding tests

[open all](#) | [close all](#)

- [Can you write code? \(](#)
- [♦ Complete the given](#)
- [Converting from A to I](#)
- [Designing your classe](#)
- [Java Data Structures](#)
- [Passing the unit tests](#)
- [What is wrong with th](#)
- [Writing Code Home A](#)
- [Written Test Core Jav](#)
- [Written Test JEE \(1\)](#)

## How good are your .....?

[open all](#) | [close all](#)

- [Career Making Know-](#)
- [Job Hunting & Resum](#)

The Fork/Join Framework in Java 7 is designed for work that can be broken down into smaller tasks and the results of those tasks combined to produce the final result. Multicore processors are now widespread across server, desktop, and laptop hardware. They are also making their way into smaller devices, such as smartphones and tablets. Fork/Join offers serious gains for solving problems that involve **recursion**. Because recursion is fundamental to parallel programming on multicore platforms.

The fork/join tasks should operate as “pure” in-memory algorithms in which no I/O operations come into play. Also, communication between tasks through shared state should be avoided as much as possible, because that implies that locking might have to be performed. Ideally, tasks communicate only when one task forks another or when one task joins another.

*ExecutorService* continues to be a fine solution for many concurrent programming tasks, and in programming scenarios in which recursion is vital to processing power, it makes sense to use Fork/join. This fork and join feature is used in Java 8 parallel stream processing with lambda expressions.

## Popular Posts

♦ [11 Spring boot interview questions & answers](#)

825 views

♦ [Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers](#)

765 views

[18 Java scenarios based interview Questions and Answers](#)

399 views

001A: ♦ [7+ Java integration styles & patterns interview questions & answers](#)

388 views

01b: ♦ [13 Spring basics Q8 – Q13 interview questions & answers](#)

295 views

## ♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

285 views

## ♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

239 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 views

Bio

Latest Posts



### Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via [Amazon.com](http://Amazon.com) in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site. **945+** paid members. [join my LinkedIn Group](#). [Reviews](#)



### About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via [Amazon.com](http://Amazon.com) in 2005, and sold 35,000+ copies. Books are outdated and replaced with

this subscription based site. **945+** paid members. [join my LinkedIn Group](#). [Reviews](#)

< ♥♦ 01: 30+ Writing low latency applications in Java interview Q&As

07: ♦ Java 8 API examples using lambda expressions and functional programming >

**Posted in** Java 7, member-paid

## Leave a Reply

Logged in as geethika. [Log out?](#)

### Comment

**Post Comment**

## Empowers you to open more doors, and fast-track

### Technical Know Hows

☀ [Java generics in no time](#) ☀ [Top 6 tips to transforming your thinking from OOP to FP](#) ☀ [How does a HashMap internally work? What is a hashing function?](#)

☀ [10+ Java String class interview Q&As](#) ☀ [Java auto un/boxing benefits & caveats](#) ☀ [Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect](#)

### Non-Technical Know Hows

☀ [6 Aspects that can motivate you to fast-track your career & go places](#) ☀ [Are you reinventing yourself as a Java developer?](#) ☀ [8 tips to safeguard your Java career against offshoring](#) ☀ [My top 5 career mistakes](#)

## Prepare to succeed

☀ [Turn readers of your Java CV go from “Blah blah” to “Wow”?](#) ☀ [How to prepare for Java job interviews?](#) ☀ [16 Technical Key Areas](#) ☀ [How to choose from multiple Java job offers?](#)

Select Category ▼

## © Disclaimer

The contents in this Java-Success are copy righted. The author has the right to correct or enhance the current content without any prior notice.

These are general advice only, and one needs to take his/her own circumstances into consideration. The author will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. No guarantees are made regarding the accuracy or usefulness of content, though I do make an effort to be accurate. Links to external sites do not imply endorsement of the linked-to sites.