

Understand Java Parallel Streams Internals: Demo' ing How to Configure the Common Fork-Join Pool

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

**Institute for Software
Integrated Systems**

**Vanderbilt University
Nashville, Tennessee, USA**



Learning Objectives in this Part of the Lesson

- Understand parallel stream internals, e.g.
 - Know what can change & what can't
 - Partition a data source into "chunks"
 - Process chunks in parallel via the common fork-join pool
- Configure the Java parallel stream common fork-join pool
 - Know the performance impact of configuring the common fork-join pool size

Entering the test program with 12 cores
ex20: testDefaultDownloadBehavior() downloaded
and stored 42 images using 12 threads
in the pool

ex20: testAdaptiveMBDownloadBehavior()
downloaded and stored 42 images using
43 threads in the pool

ex20: testAdaptiveBTDownloadBehavior()
downloaded and stored 42 images using
43 threads in the pool

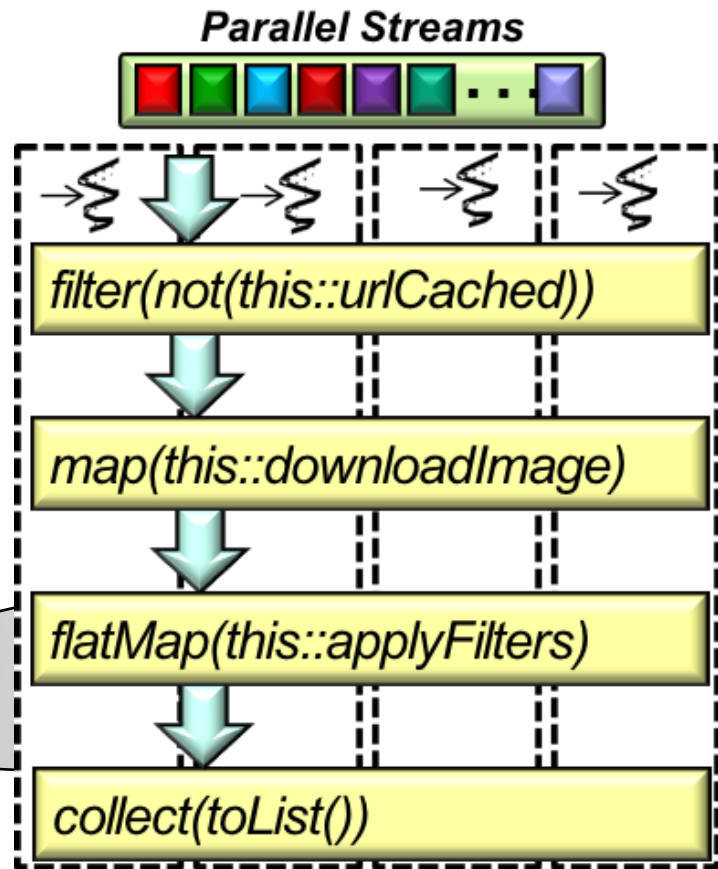
Printing 3 results from fastest to slowest
testAdaptiveBTDownloadBehavior() executed in
3598 msecs
testAdaptiveMBDownloadBehavior() executed in
3910 msecs
testDefaultDownloadBehavior() executed in
4104 msecs

Leaving the test program

Demo'ing Impact of Configuring Common Fork-Join Pool

Demo'ing Impact of Configuring Common Fork-Join Pool

- The common fork-join pool size can be controlled programmatically



See prior lesson on *"Java Parallel Stream Internals: Configuring the Common Fork-Join Pool"*

Demo'ing Impact of Configuring Common Fork-Join Pool

- The common fork-join pool size can be controlled programmatically
 - This demo applies the Managed Blocker interface to adaptively add new worker threads to the Java common fork-join pool

```
File downloadAndStoreImageMB
```

```
(URL url) {  
    final Image[] image =  
        new Image[1];  
    ...
```

```
ForkJoinPool
```

```
    .managedBlock(new ForkJoinPool  
        .ManagedBlocker() {  
            public boolean block() {  
                image[0] =  
                    downloadImage(url);  
                return true;  
            } ... });
```

```
return image[0].store(); ...
```



Demo'ing Impact of Configuring Common Fork-Join Pool

- This program shows the performance difference of using ManagedBlocker versus not using ManagedBlocker for an I/O-intensive app

```
void testDownloadBehavior(Function<URL, File>
                           downloadAndStoreImage,
                           String testName) {
    ...
    List<File> imageFiles = Options.instance()
        .getUrlList()
        .parallelStream()

        .map(downloadAndStoreImage)

        .collect(Collectors.toList());
    printStats(testName, imageFiles.size()); ...
}
```

Demo'ing Impact of Configuring Common Fork-Join Pool

- This program shows the performance difference of using ManagedBlocker versus not using ManagedBlocker for an I/O-intensive app

```
void testDownloadBehavior(Function<URL, File>
                          downloadAndStoreImage,
                          String testName) {
    ...
    List<File> imageFiles = Options.instance()
        .getUrlList()
        .parallelStream()
        .map(downloadAndStoreImage)

        .collect(Collectors.toList());
    printStats(testName, imageFiles.size()); ...
}
```

This function param is used to pass different strategies for downloading & storing images from remote websites

Demo'ing Impact of Configuring Common Fork-Join Pool

- Results show increasing worker threads in the pool improves performance

Entering the test program with 12 cores

ex20: testDefaultDownloadBehavior() downloaded and stored 42 images
using 12 threads in the pool

ex20: testAdaptiveMBDownloadBehavior() downloaded and stored 42 images
using 43 threads in the pool

ex20: testAdaptiveBTDownloadBehavior() downloaded and stored 42 images
using 43 threads in the pool

Printing 3 results from fastest to slowest

testAdaptiveBTDownloadBehavior() executed in 3598 msecs

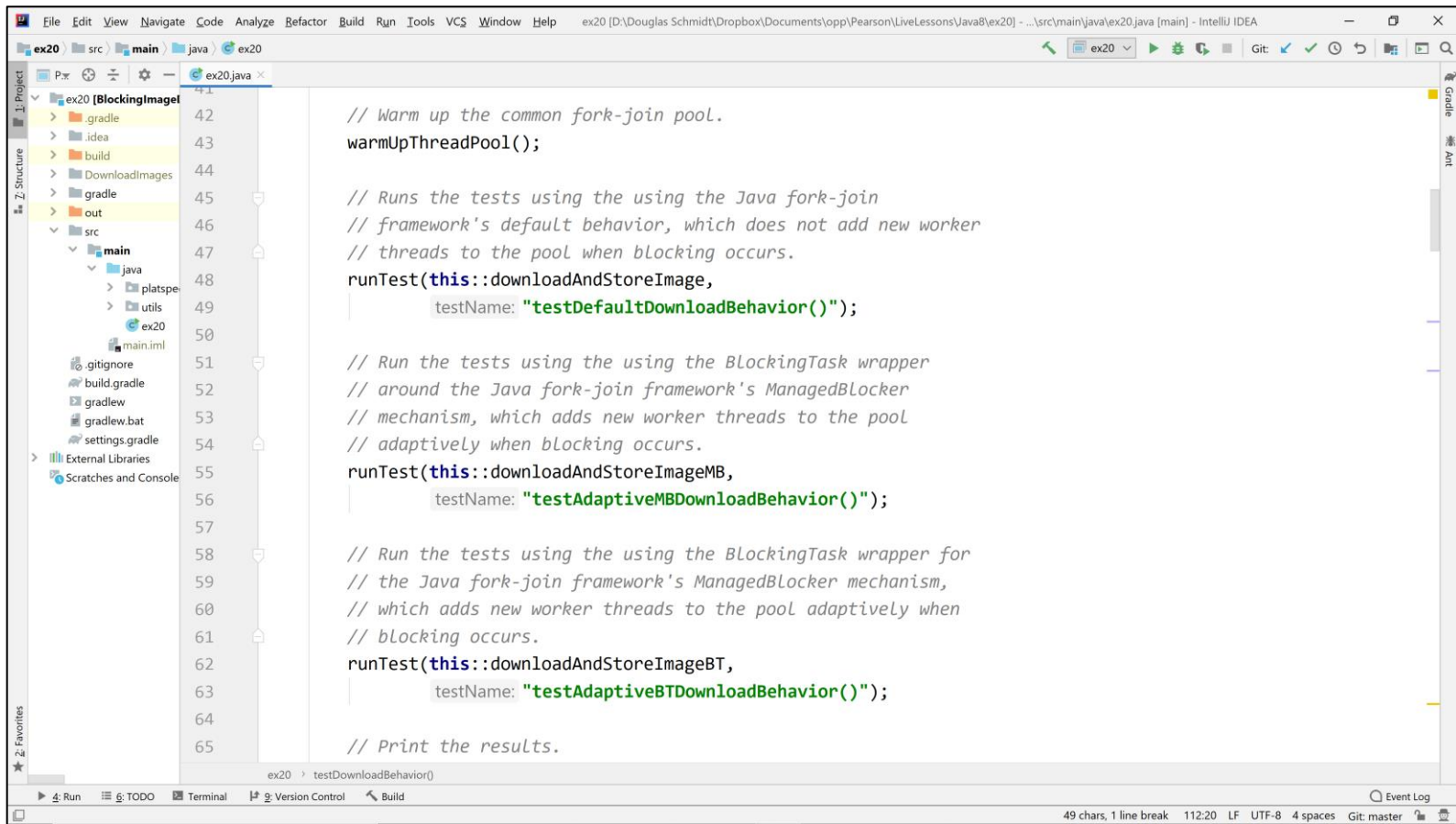
testAdaptiveMBDownloadBehavior() executed in 3910 msecs

testDefaultDownloadBehavior() executed in 4104 msecs

Leaving the test program

See upcoming lessons on "*The Java Fork-Join Pool: the ManagedBlocker Interface*"

Demo'ing Impact of Configuring Common Fork-Join Pool



```
42 // Warm up the common fork-join pool.
43 warmUpThreadPool();
44
45 // Runs the tests using the using the Java fork-join
46 // framework's default behavior, which does not add new worker
47 // threads to the pool when blocking occurs.
48 runTest(this::downloadAndStoreImage,
49         testName: "testDefaultDownloadBehavior()");
50
51 // Run the tests using the using the BlockingTask wrapper
52 // around the Java fork-join framework's ManagedBlocker
53 // mechanism, which adds new worker threads to the pool
54 // adaptively when blocking occurs.
55 runTest(this::downloadAndStoreImageMB,
56         testName: "testAdaptiveMBDownloadBehavior()");
57
58 // Run the tests using the using the BlockingTask wrapper for
59 // the Java fork-join framework's ManagedBlocker mechanism,
60 // which adds new worker threads to the pool adaptively when
61 // blocking occurs.
62 runTest(this::downloadAndStoreImageBT,
63         testName: "testAdaptiveBTDownloadBehavior()");
64
65 // Print the results.
```

The screenshot shows the IntelliJ IDEA IDE with a project named 'ex20'. The left sidebar displays the project structure, including 'src/main/java/ex20'. The main editor window shows the 'ex20.java' file with the code above. The status bar at the bottom indicates '49 chars, 1 line break 112:20 LF UTF-8 4 spaces Git: master'.

See github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex20

End of Understand Java Parallel Streams Internals: Demo'ing How to Configure the Common Fork-Join Pool