CSCI 4401

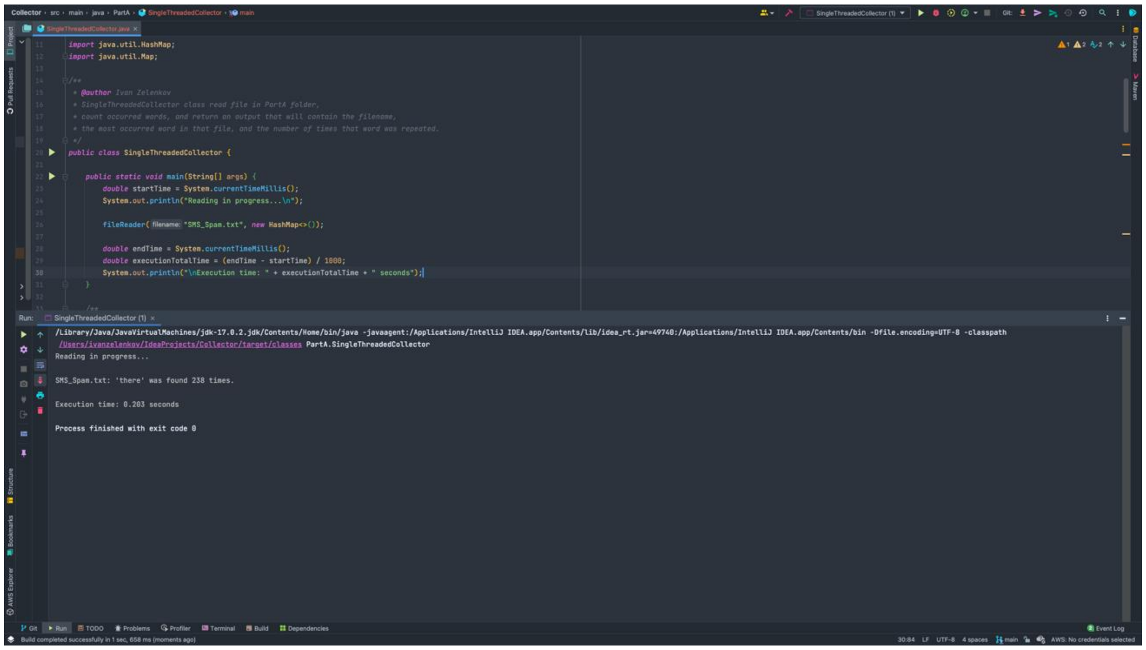
Dr. James Wagner

Ivan Zelenkov

30 October 2022

Observations

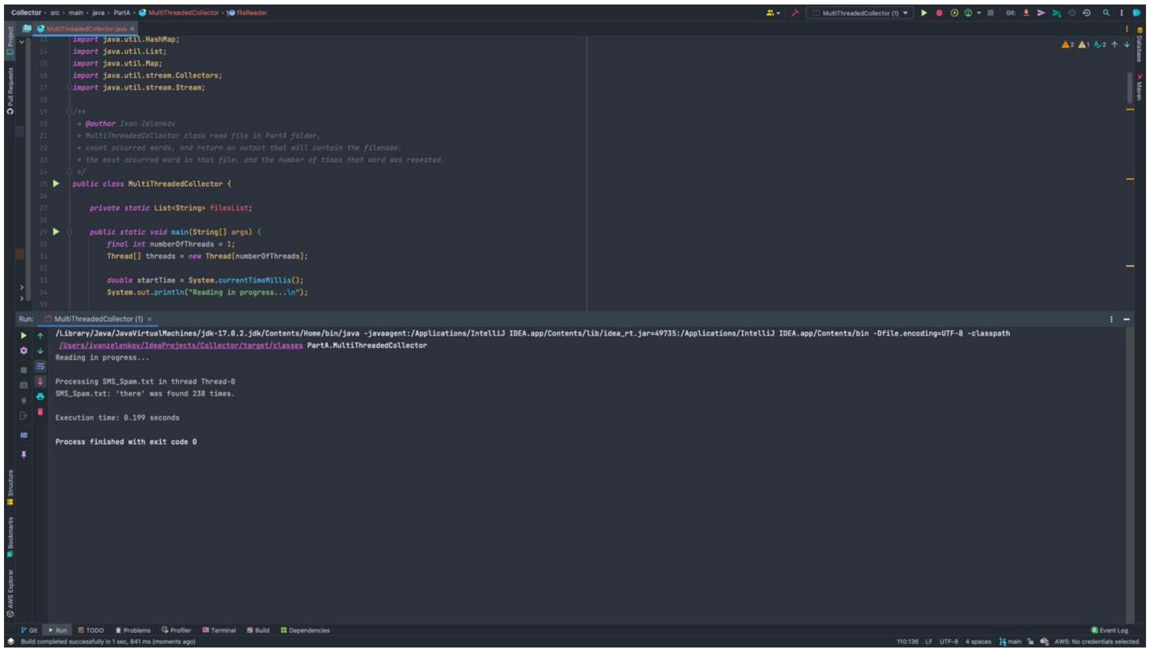
PartA.1



```java
import java.util.HashMap;
import java.util.Map;

/**
 * @author Ivan Zelenkov
 * SingleThreadedCollector class read file in PartA folder,
 * count occurred words, and return an output that will contain the filename,
 * the most occurred word in that file, and the number of times that word was repeated.
 */
public class SingleThreadedCollector {

    public static void main(String[] args) {
        double startTime = System.currentTimeMillis();
        System.out.println("Reading in progress...\n");

        fileReader( filename: "SMS_Spam.txt", new HashMap<>());

        double endTime = System.currentTimeMillis();
        double executionTotalTime = (endTime - startTime) / 1000;
        System.out.println("\nExecution time: " + executionTotalTime + " seconds");
    }
```

```
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=49740:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath
 /Users/ivanzelenkov/IdeaProjects/Collector/target/classes PartA.SingleThreadedCollector
Reading in progress...

SMS_Spam.txt: 'there' was found 238 times.

Execution time: 0.203 seconds

Process finished with exit code 0
```

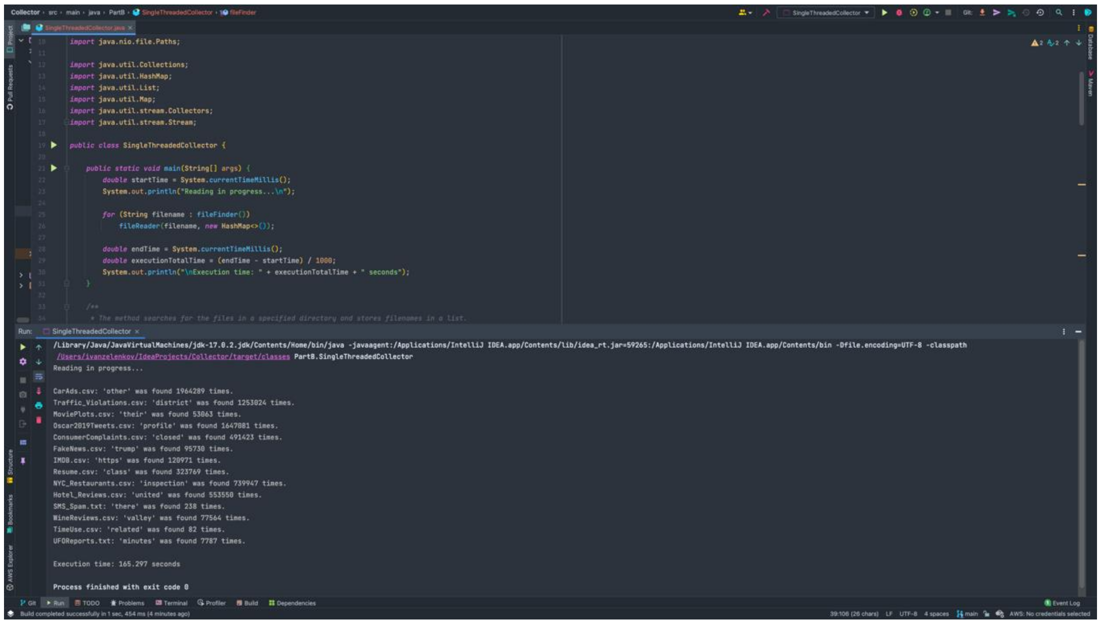
PartA.2



```java
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;
import java.util.stream.Stream;

/**
 * @author Ivan Zelenkov
 * MultiThreadedCollector class read file in PartA folder,
 * count occurred words, and return an output that will contain the filename,
 * the most occurred word in that file, and the number of times that word was repeated.
 */
public class MultiThreadedCollector {

    private static List<String> filesList;

    public static void main(String[] args) {
        final int numberOfThreads = 1;
        Thread[] threads = new Thread[numberOfThreads];

        double startTime = System.currentTimeMillis();
        System.out.println("Reading in progress...\n");
```

```
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=49735:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath
 /Users/ivanzelenkov/IdeaProjects/Collector/target/classes PartA.MultiThreadedCollector
Reading in progress...

Processing SMS_Spam.txt in thread Thread-0
SMS_Spam.txt: 'there' was found 238 times.

Execution time: 0.199 seconds

Process finished with exit code 0
```

PartB.3

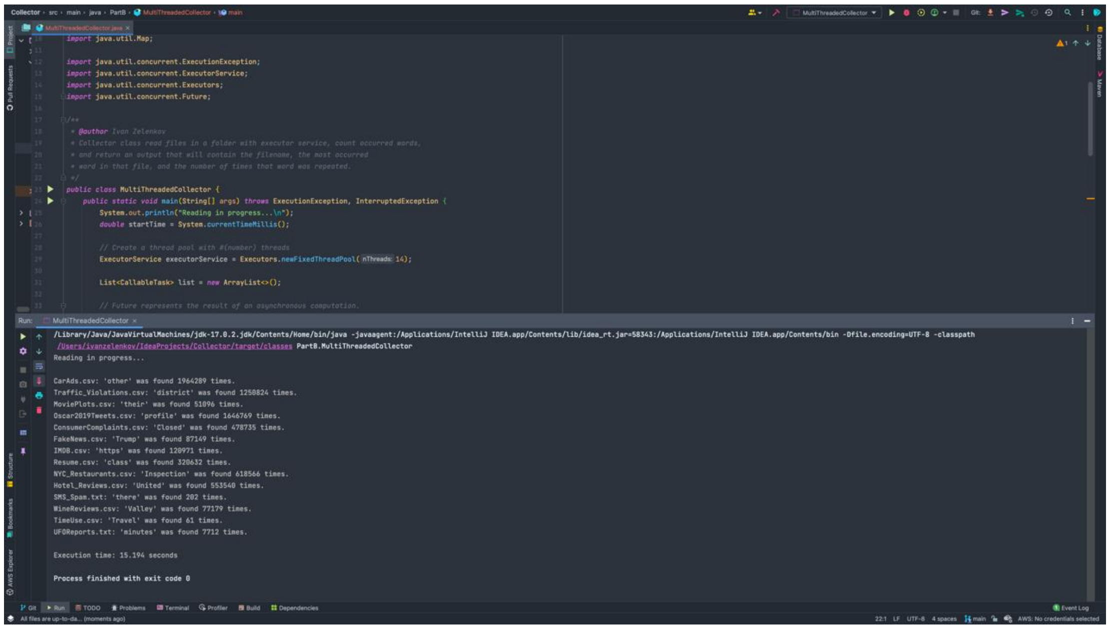


```java
import java.nio.file.Paths;

import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class SingleThreadedCollector {

    public static void main(String[] args) {
        double startTime = System.currentTimeMillis();
        System.out.println("Reading in progress...\n");

        for (String filename : fileFinder())
            fileReader(filename, new HashMap<>());

        double endTime = System.currentTimeMillis();
        double executionTotalTime = (endTime - startTime) / 1000;
        System.out.println("\nExecution time: " + executionTotalTime + " seconds");
    }

    /**
     * The method searches for the files in a specified directory and stores filenames in a list.
```

```
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=59265:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath
 /Users/ivanzelenkov/IdeaProjects/Collector/target/classes PartB.SingleThreadedCollector
Reading in progress...

CarAds.csv: 'other' was found 1964289 times.
Traffic_Violations.csv: 'district' was found 1253024 times.
MoviePlots.csv: 'their' was found 53063 times.
Oscar2019Tweets.csv: 'profile' was found 1647081 times.
ConsumerComplaints.csv: 'closed' was found 491423 times.
FakeNews.csv: 'trump' was found 95730 times.
IMDB.csv: 'https' was found 120971 times.
Resume.csv: 'class' was found 323769 times.
NYC_Restaurants.csv: 'inspection' was found 739947 times.
Hotel_Reviews.csv: 'united' was found 553550 times.
SMS_Spam.txt: 'there' was found 238 times.
WineReviews.csv: 'valley' was found 77564 times.
TimeUse.csv: 'related' was found 82 times.
UFOReports.txt: 'minutes' was found 7787 times.

Execution time: 165.297 seconds

Process finished with exit code 0
```

Collector › src › main › java › PartB › SingleThreadedCollector › fileFinder
SingleThreadedCollector.java
Run: SingleThreadedCollector
Git Run TODO Problems Terminal Profiler Build Dependencies Event Log
Build completed successfully in 1 sec, 454 ms (4 minutes ago)
39:106 (26 chars) LF UTF-8 4 spaces main AWS: No credentials selected

PartB.4



```java
import java.util.Map;

import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

/**
 * @author Ivan Zelenkov
 * Collector class read files in a folder with executor service, count occurred words,
 * and return an output that will contain the filename, the most occurred
 * word in that file, and the number of times that word was repeated.
 */
public class MultiThreadedCollector {
    public static void main(String[] args) throws ExecutionException, InterruptedException {
        System.out.println("Reading in progress...\n");
        double startTime = System.currentTimeMillis();

        // Create a thread pool with #(number) threads
        ExecutorService executorService = Executors.newFixedThreadPool( nThreads: 14);

        List<CallableTask> list = new ArrayList<>();

        // Future represents the result of an asynchronous computation.
```

```
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=58343:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath
 /Users/ivanzelenkov/IdeaProjects/Collector/target/classes PartB.MultiThreadedCollector
Reading in progress...

CarAds.csv: 'other' was found 1964289 times.
Traffic_Violations.csv: 'district' was found 1250824 times.
MoviePlots.csv: 'their' was found 51096 times.
Oscar2019Tweets.csv: 'profile' was found 1646769 times.
ConsumerComplaints.csv: 'Closed' was found 478735 times.
FakeNews.csv: 'Trump' was found 87149 times.
IMDB.csv: 'https' was found 120971 times.
Resume.csv: 'class' was found 320632 times.
NYC_Restaurants.csv: 'Inspection' was found 618566 times.
Hotel_Reviews.csv: 'United' was found 553540 times.
SMS_Spam.txt: 'there' was found 202 times.
WineReviews.csv: 'Valley' was found 77179 times.
TimeUse.csv: 'Travel' was found 61 times.
UFOReports.txt: 'minutes' was found 7712 times.

Execution time: 15.194 seconds

Process finished with exit code 0
```

Collector › src › main › java › PartB › MultiThreadedCollector › main
MultiThreadedCollector.java
Run: MultiThreadedCollector
Git Run TODO Problems Terminal Profiler Build Dependencies Event Log
All files are up-to-da... (moments ago)
22:1 LF UTF-8 4 spaces main AWS: No credentials selected

Was there a difference in your runtimes between Part B.3 and Part B.4? Why or why not?

The runtime of the single-threaded version (PartB.3) was 165.297 seconds, and the multithreaded version using 14 threads (PartB.4) was 15.194 seconds. From this data, I can infer that the multithreaded version is 10.879 times faster than the single-threaded version.

The implementations are different and as we can see the second multi-threaded version (PartB.4) is much faster because I used an Executor with a thread pool. There was no good result when I was using Thread class (PartB.3) because it did not give me the speed I wanted and was very cumbersome as the opposite of using Executor and implementing the Callable interface.

Creating a Thread object, starting threads one by one where each of them will call the run() method is not a good way to work with multithreading in Java nowadays. Instead, Oracle simplified things for us by providing an Executor that can process tasks asynchronously without having to deal with threads directly.

I was able to test the new multithreaded version with any number of threads. All I had to do is change an argument when calling the newFixedThreadPool method of class Executors.

Number of Threads	Execution Time (sec)	Comparison with the previous implementation of the single-threaded version (165.297 sec) (n times faster)
1	60.065	2.750
2	34.474	4.794
3	36.011	4.590
4	36.902	4.479
5	40.185	4.113
6	53.091	3.113
7	44.779	3.691
8	40.603	4.071
9	55.315	2.988
10	59.663	2.770

11	17. 664	9. 357
12	24. 649	6. 706
13	22. 566	7. 325
14	15. 194	10. 879

Based on the table above, we can see that a multithreaded program with 14 threads runs in 15.194 seconds, which is the fastest execution time I got. If we run this program with only 1 thread, then it will give us 60.065 seconds, but it is still much faster than the single-threaded version with a different implementation by 2.75 times.

Was there a difference in your output between Part B.3 and Part B.4? Why or why not?

When I was working with multithreaded version that was using Thread class, I had a different output comparing to the single-threaded version. Single-threaded program returns result about files as they are stored and processed in alphabetical order (sequentially). The output of the first multi-threaded version was different because it was returning data about each file when some thread finished reading it first.

The implementation that uses Executor returns the output the same as a single-threaded program. The tasks are added to the list of callable tasks where each of them represents an object of CallableTask class. That class implements a Callable interface where I implemented a required call() method that will be called when the task is submitted by the executor.

```
callableTasks.add(new CallableTask(f.getAbsolutePath()));
```

After that, they are submitted to the result list in the same order,

```
result.add(executorService.submit(callableTask));
```

processed asynchronously and returned to result list as Future objects that contain the result of computations. All that we have left to do is to iterate through Future objects as they are stored in a result list and output the result of their computations to the user’s screen.

In conclusion, it was very exciting to look at that problem from different sides and explore ways to solve it. As was observed both multithreaded programs are faster than single-threaded programs. But the traditional multithreaded approach has a lot of redundant code which causes more blocks of code to iterate and execute while using an Executor we can get data from

14 files in 15.194 seconds, and we can assume that on average the multithreaded program spends about 1.085 seconds per file.

```
CarAds.csv: 'other' was found 1964289 times.
Traffic_Violations.csv: 'district' was found 1253020 times.
MoviePlots.csv: 'their' was found 52952 times.
Oscar2019Tweets.csv: 'profile' was found 1646901 times.
ConsumerComplaints.csv: 'closed' was found 487805 times.
FakeNews.csv: 'trump' was found 89279 times.
IMDB.csv: 'https' was found 120971 times.
Resume.csv: 'class' was found 320978 times.
NYC_Restaurants.csv: 'inspection' was found 739947 times.
Hotel_Reviews.csv: 'united' was found 553549 times.
SMS_Spam.txt: 'there' was found 233 times.
WineReviews.csv: 'valley' was found 77460 times.
TimeUse.csv: 'related' was found 82 times.
UFOReports.txt: 'minutes' was found 7785 times.
```