Michael Weaver, Caleb Miller, Mathew Paschall

CST – 235

Brandon Bass

12/17/17

**CLC 6**

GitHub Link: <https://github.com/battousairurik/CST-235>

**Assignment Approach**

1. For this assignment I followed an example that I found online. I use the concurrent Executor Service to utilize several threads to solve the problem of finding the sum of the work. Each recursive call breaks the work down into smaller sections until a work minimum is met and then the sum generated. For ease, the steps are written to the console.
2. The second half of the assignment calls for factors that exist that slow down java based applications. Then once the best three factors are picked, finding if those factors can be alleviated by the use of parallel programming.

**Questions**

1. The fork/join framework is an implementation of the Executor Service interface which aims to make use of multiple processors. It breaks down work into smaller portions recursively and assigns each smaller portion to a different processor to enhance processing power and performance. The service distributes work between a pool of threads, and unique to the fork/join framework is the work stealing algorithm which allows portions of work to be taken by inactive threads from other threads that are still active (Oracle, 2017). Specifically, Fork is when the work is broken down and passed into different processors. Join is when the work is merged and passed back up the chain to be processed for some final solution.
2. There are many factors that can hinder the execution of an application, in my mind, the most important and obvious would be hardware.

* Hardware is not unlimited even when using the cloud and dynamically expanding hardware, once the CPU hits its maximum load, the application suffers. Magically adding hardware out of thin air is impossible, but accessing 100% of the hardware available is key when optimizing applications. This is where fork/join comes into play to task multiple processes to execute simultaneously.
* Logging is an exhausting feature. Logging uses up power, processes, and memory. While debug logging will help in certain situations, in production environments, keep the logging to a minimum level (perhaps ‘INFO’ or ‘WARN’ log4j levels). An additional side effect of excessive logging is Disk drives filling up, which can have its own consequences. (Karun Subramanian. 2014.) Logging can be minimized by parallel programming with both reading and writing.
* (Mathew’s input)

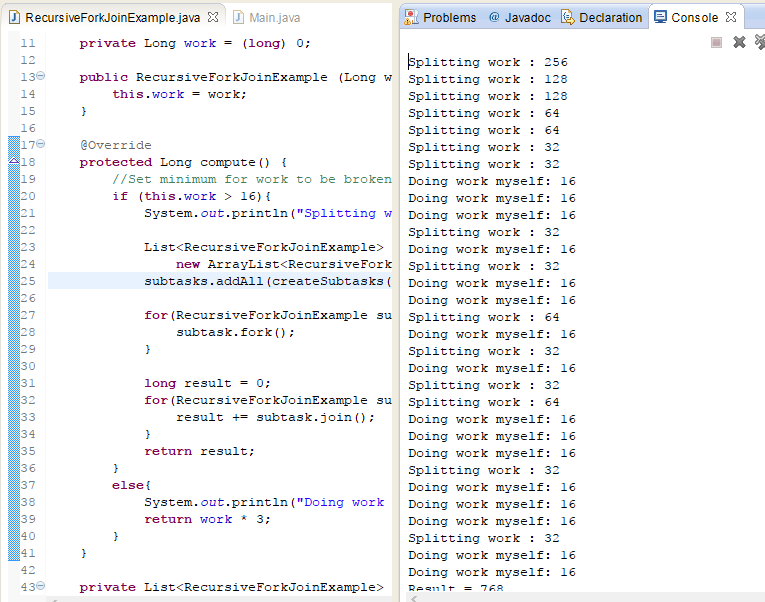
**List of Classes, Methods, Variables, etc.**

*Fork/Join Example*

RecursiveForkJoinExample.java (containing work field, constructor, compute, and create subtasks methods) and Main.java (containing only the main method to run an example).

**Screenshots**

Fork/Join Example



Logging

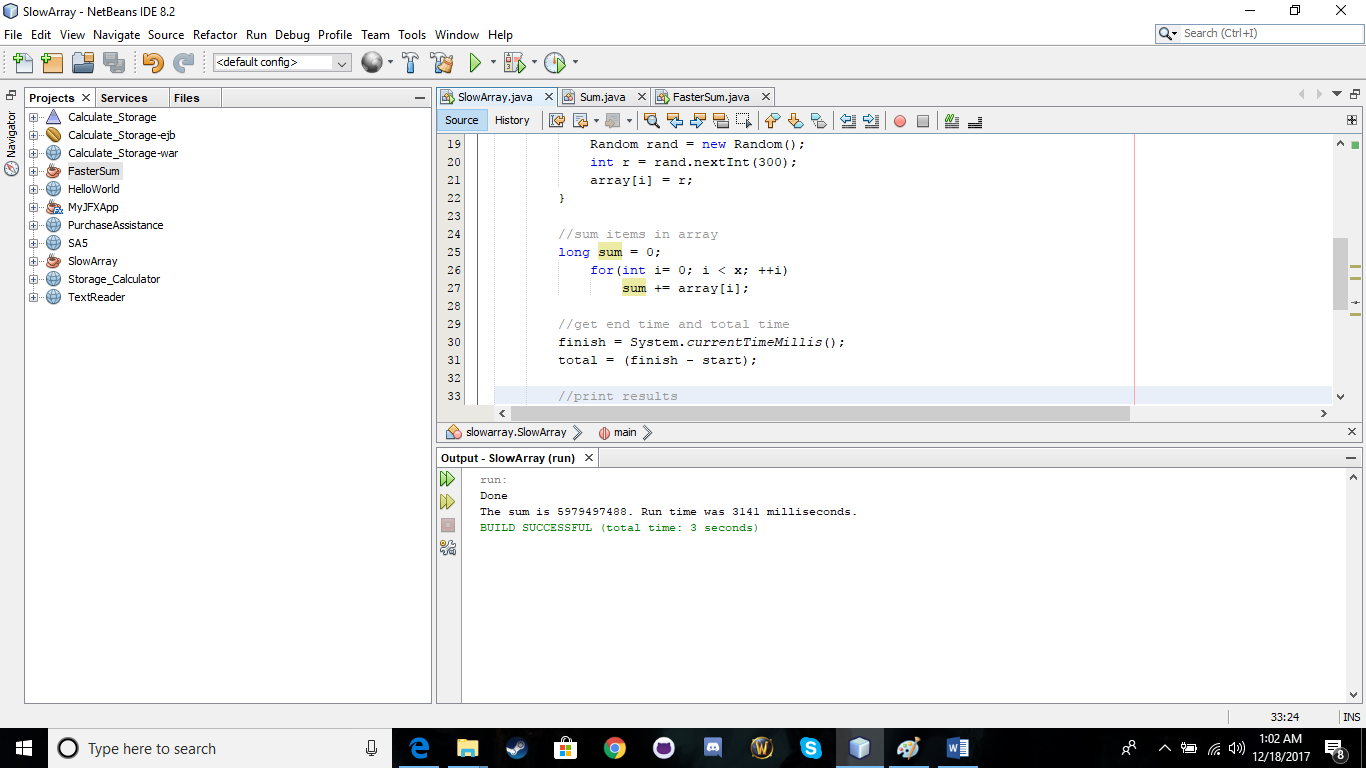


**Mathew**

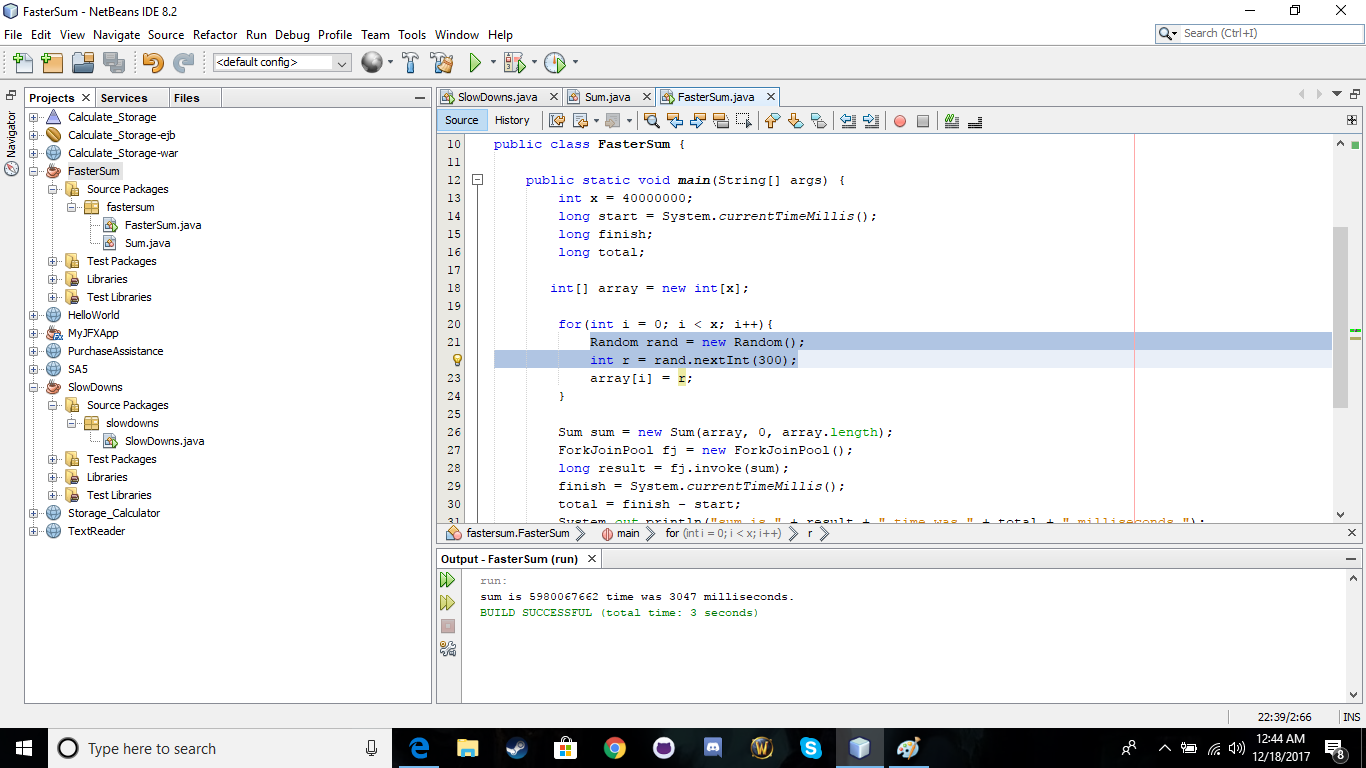
Approach:

I created a very large array of random integers and had the program add them all together. I had some trouble using the fork method to make this work properly, but after a lot of research I got it to work properly. I think with more computations the fork method would speed up processes with arrays, especially sorting. I also think that loading images would be a good thing to use parallel programming to achieve.

Array Sum Slowdown



Faster Array Sum



**References**

Oracle. (2017). *Fork/Join*. Retrieved from <https://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html>

Jenkov, Jacob. (2015). *Java Fork and Join using ForkJoinPool.* Retrieved from <http://tutorials.jenkov.com/java-util-concurrent/java-fork-and-join-forkjoinpool.html>

Subramanian, Karun. (2014). *Top 10 Reasons why your Enterprise Application is slow.* Retrieved from: <http://karunsubramanian.com/websphere/top-10-reasons-why-your-enterprise-java-application-is-slow/>