Software Engineering
Final Project

(This is now the final version of the large project requirements)

The overall project will be graded out of 100 points. Most (though not all) of the pieces are from the weekly assignments, so make sure to incorporate any feedback from those to get everything in great shape.

Functionality Requirements:

Project Infrastructure: Assignment 1
- A repository on github, with **branch protection** for 'main' requiring pull requests with approvals and passing status checks
- A checkstyle status check that verifies everything in **checkstyle_checks.xml** from Brightspace plus at least one **additional** check
- (Assignment 5) a gradle status check that verifies that all the code **compiles** and all **tests pass**

Documentation and Design: Assignment 2
- A README with a system design diagram and details of the computation
- 3 interface classes for APIs between different components of the system

Testing: Assignment 3
- At least one unit test for each component
- One integration test verifying that the coordination and compute engine pieces work together
- (Assignment 5) TestMultiUser to verify that multiple users can make requests at the same time
- Note: you will probably want additional tests to be confident in the overall functionality

Working Implementation: Assignment 4
- The engine should successfully compute the correct output for any given integer greater than zero
- (**New**) The engine should gracefully handle any invalid input (return an error to the user, adapt the computation to apply to the input, or log an error to the output file = graceful handling; crashing/failing silently = not graceful)

- (**New**) The engine should accept **arbitrarily large** inputs
- (Assignment 6) There should be a client of some sort that can be used to make requests of the engine

Multithreading the engine: Assignment 5
- Multiple users should be able to make simultaneous requests to the engine
- The compute engine should be able to use multiple threads to compute the results more quickly than if it were running on a single CPU

Network Calls: Assignment 6
- Support cross-network calls with grpc between the client/end user and the computation coordination layer
- Support cross-network calls with grpc between the computation engine and the data store layer.

Performance Tuning: Assignment 7
- Have a documented performance improvement, including a benchmark test, benchmark performance numbers, and what the improvement was. You do not need to keep the old slow version around in the code base if you'd rather clean it up.

Something Extra (New)

You've built a cool system, now have some fun with it! Pick one of the previous assignments, or an entirely different topic, and go further with it. Some examples (you're welcome to use one of these, or pick your own):
- Investigate how to start up servers from within gradle, and use that to create an automated end-to-end testing of the grpc logic that runs along with the rest of the tests in github for every pull request
- Create a fancier client that uses the cross-language abilities of grpc to connect a non-Java client (such as javascript in a web browser) to the Java server
- Set up producer/consumer multithreading to parallelize I/O as well as CPU multithreading

- Extend the output of the engine to not just be text; create useful visualizations for the user either as a per-input computation or to summarize the computation job, and display those after a successful computation.
- Something else! If you aren't sure if you've picked something substantial enough, feel free to email me your ideas to double-check.

If you've already done something earlier in the semester that falls into this category, you're free to re-use it here!

Process Requirements

As always, any code changes should have nice descriptive commit messages and reviewed pull requests that pass all status checks

Conduct and Document a 5 Why's Restrospective

Using the 5 Why's approach we covered in class (much like Agile programming, there are many variants of this if you poke around online):

- Pick something from the project that didn't go perfectly (doesn't have to have gone 'wrong' exactly, just something less than optimal).
- Go through the 5 Why's process to come up with at least one **process change** that would improve how this problem would go if it happened again
- Document what the "whys" were (remember, always have one more "why" past the actionable one!) in a **text document**, and include that document in a 'documentation' **folder** in your repo. This can be a fairly concise summary - one or two sentences per question/answer pair, plus any actionable **process changes** associated with the question/answer pair (you should have at least one).
- Link this document in your README