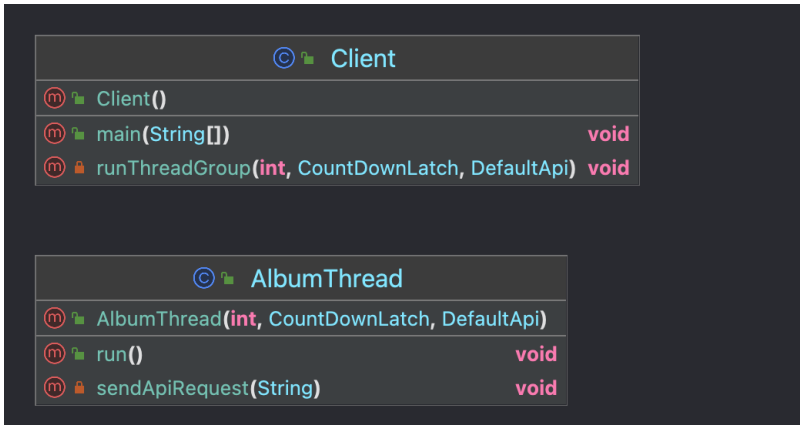
CS 6650 Assignment 1 Report

Tzu-Yu Huang

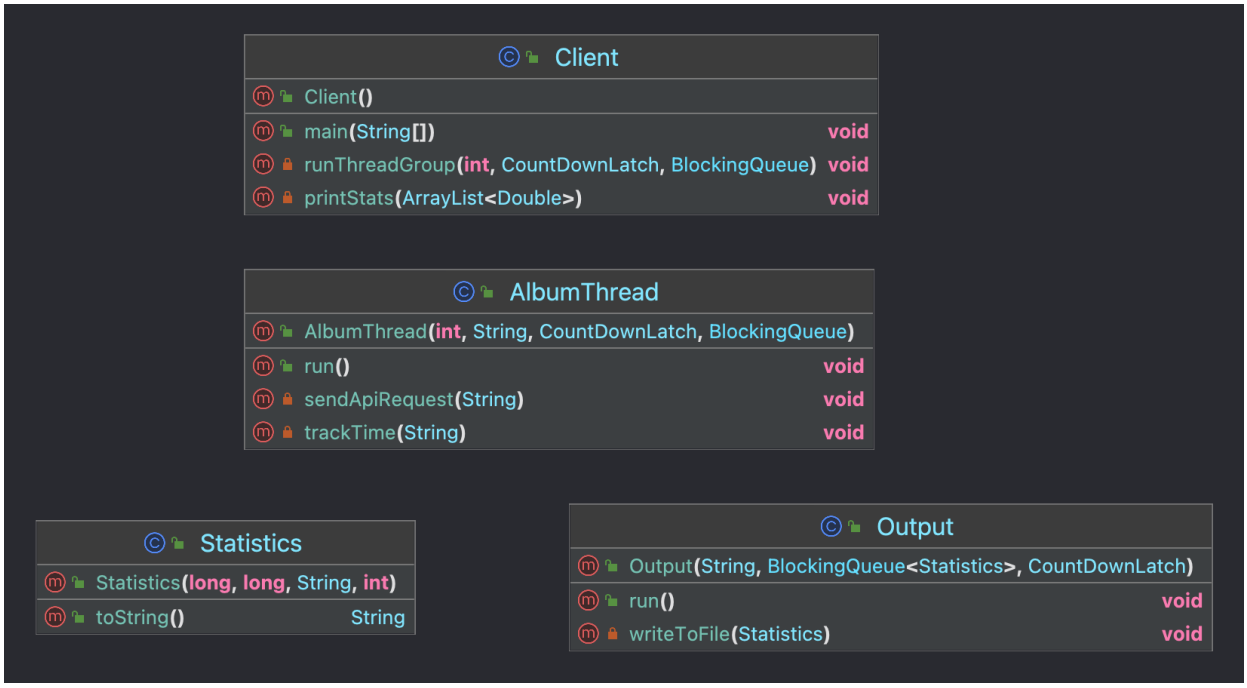
1 - Github Repo

https://github.com/tyhuang06/CS6650-assignments/tree/main/a1

2 - Client Design



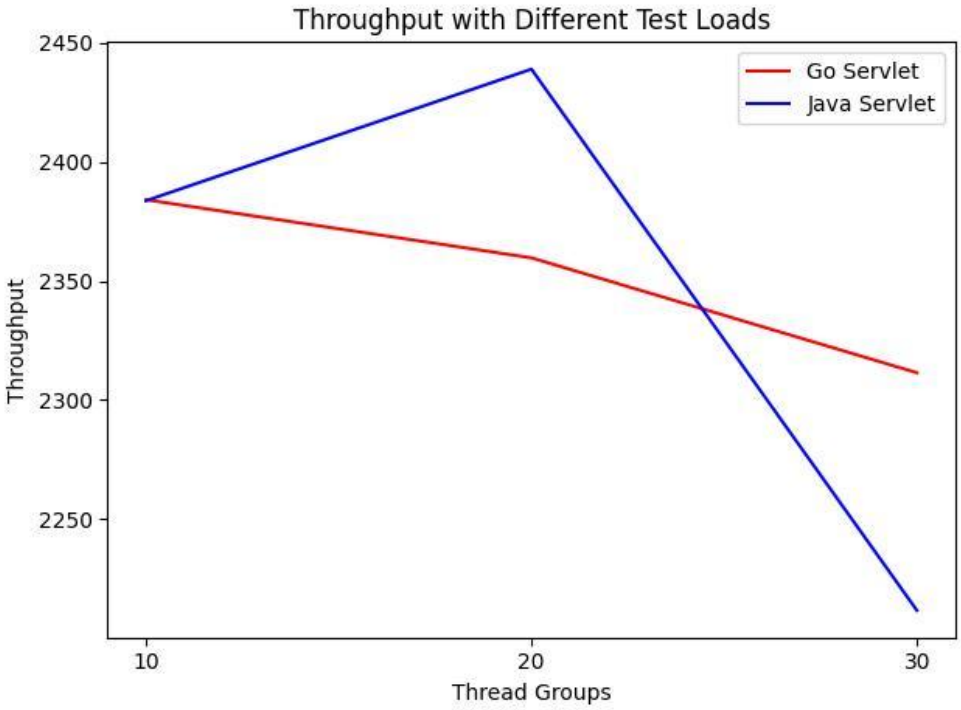
The design for part 1 client is simple. We take in parameters from the main function in Client, create a connection to the server, then start the threads we need. The AlbumThread class handles running the threads and making api requests.



For the part2 client, I utilized the Producer Consumer Pattern in order to write the result to a csv file. The class AlbumThread serves as the producer and Output serves as the client. The Statistics class is just a format wrapper for all the stats we need. The overall thread running logic is similar to the part1 design, I added some functions such as trackTime() and printStats() to calculate the additional information required.

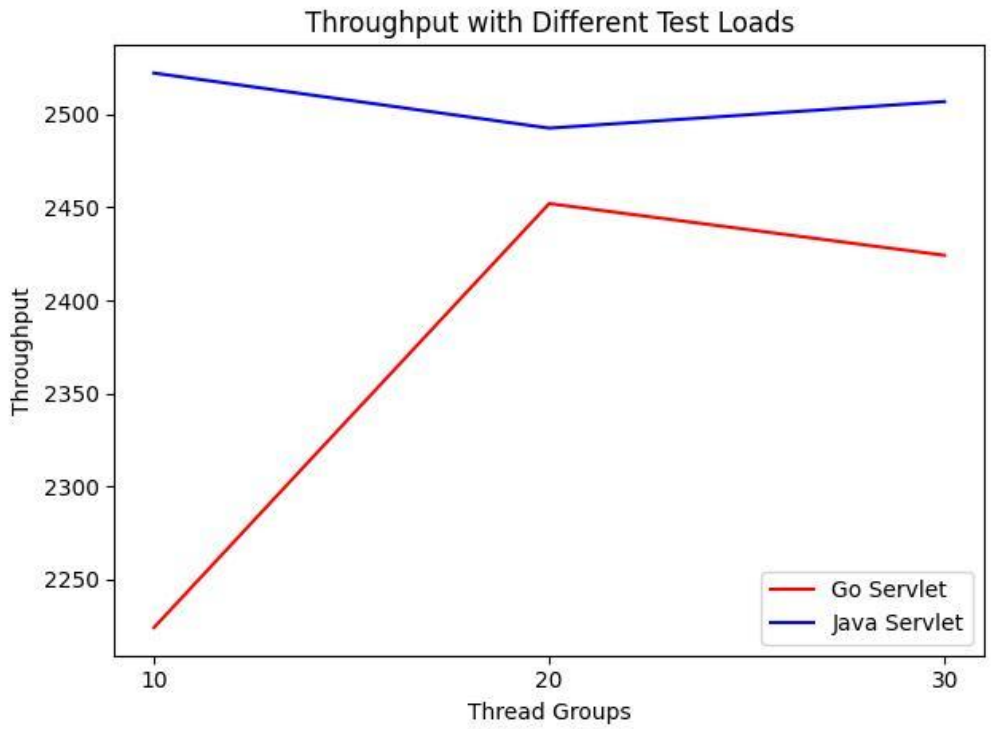
3 - Client Results (Part 1)

	Java Server	Go Server
10	Starting client with 10 thread groups of 10 threads each Wall time: 83.903 seconds Throughput: 2383.704992670107 requests/second	Starting client with 10 thread groups of 10 threads each Wall time: 83.888 seconds Throughput: 2384.131222582491 requests/second
20	Starting client with 20 thread groups of 10 threads each Wall time: 164.001 seconds Throughput: 2439.009518234645 requests/second	Starting client with 20 thread groups of 10 threads each Wall time: 169.513 seconds Throughput: 2359.701025880021 requests/second
30	Starting client with 30 thread groups of 10 threads each Wall time: 271.305 seconds Throughput: 2211.5331453530157 requests/second	Starting client with 30 thread groups of 10 threads each Wall time: 259.581 seconds Throughput: 2311.417245484068 requests/second



4 - Client Results (Part 2)

	Java Server	Go Server
10	Starting client with 10 thread groups of 10 threads each Wall time: 79.294 seconds Throughput: 2522.2589351022775 requests/second POST times: Min: 21.0 Max: 220.0 Mean: 31.3403 Median: 30.0 99th percentile: 62.0 GET times: Min: 17.0 Max: 106.0 Mean: 26.8622 Median: 26.0 99th percentile: 55.0	Starting client with 10 thread groups of 10 threads each Wall time: 89.929 seconds Throughput: 2223.9766927242604 requests/second POST times: Min: 22.0 Max: 288.0 Mean: 36.0827 Median: 33.0 99th percentile: 73.0 GET times: Min: 14.0 Max: 352.0 Mean: 31.9335 Median: 28.0 99th percentile: 72.0
20	Starting client with 20 thread groups of 10 threads each Wall time: 160.473 seconds Throughput: 2492.6311591358044 requests/second POST times: Min: 21.0 Max: 270.0 Mean: 31.54485 Median: 30.0 99th percentile: 64.0 GET times: Min: 15.0 Max: 291.0 Mean: 27.383 Median: 26.0 99th percentile: 56.0	Starting client with 20 thread groups of 10 threads each Wall time: 163.128 seconds Throughput: 2452.062184296994 requests/second POST times: Min: 20.0 Max: 216.0 Mean: 32.8258 Median: 31.0 99th percentile: 60.0 GET times: Min: 16.0 Max: 188.0 Mean: 27.80655 Median: 27.0 99th percentile: 47.0
30	Starting client with 30 thread groups of 10 threads each Wall time: 239.337 seconds Throughput: 2506.925381366024 requests/second POST times: Min: 21.0 Max: 303.0 Mean: 31.401366666666668 Median: 30.0 99th percentile: 62.0 GET times: Min: 16.0 Max: 685.0 Mean: 27.000533333333333 Median: 26.0 99th percentile: 55.0	Starting client with 30 thread groups of 10 threads each Wall time: 247.5 seconds Throughput: 2424.242424242424 requests/second POST times: Min: 21.0 Max: 308.0 Mean: 32.692033333333335 Median: 31.0 99th percentile: 60.0 GET times: Min: 15.0 Max: 323.0 Mean: 28.671466666666667 Median: 27.0 99th percentile: 51.0



5 - Single test throughput overtime

Java servlet with `threadGroupSize = 10, numThreadGroups = 30, delay = 2`