CS6650 - Assignment 1

Spring 2021 - Panpan Cao

GitHub Repo: https://github.com/PPBlackwater363/CS6650-BSDS

Client Design

The client contains 4 classes and 1 configuration file.

MultiThreadClient — The class contains the main function and takes arguments as input to create threads and executes them. Per the requirement, there are three phases in the main function. The logic of each phase is based on the number of purchases. Once the function executed all phases, the class would do the calculation and generate the csv file.

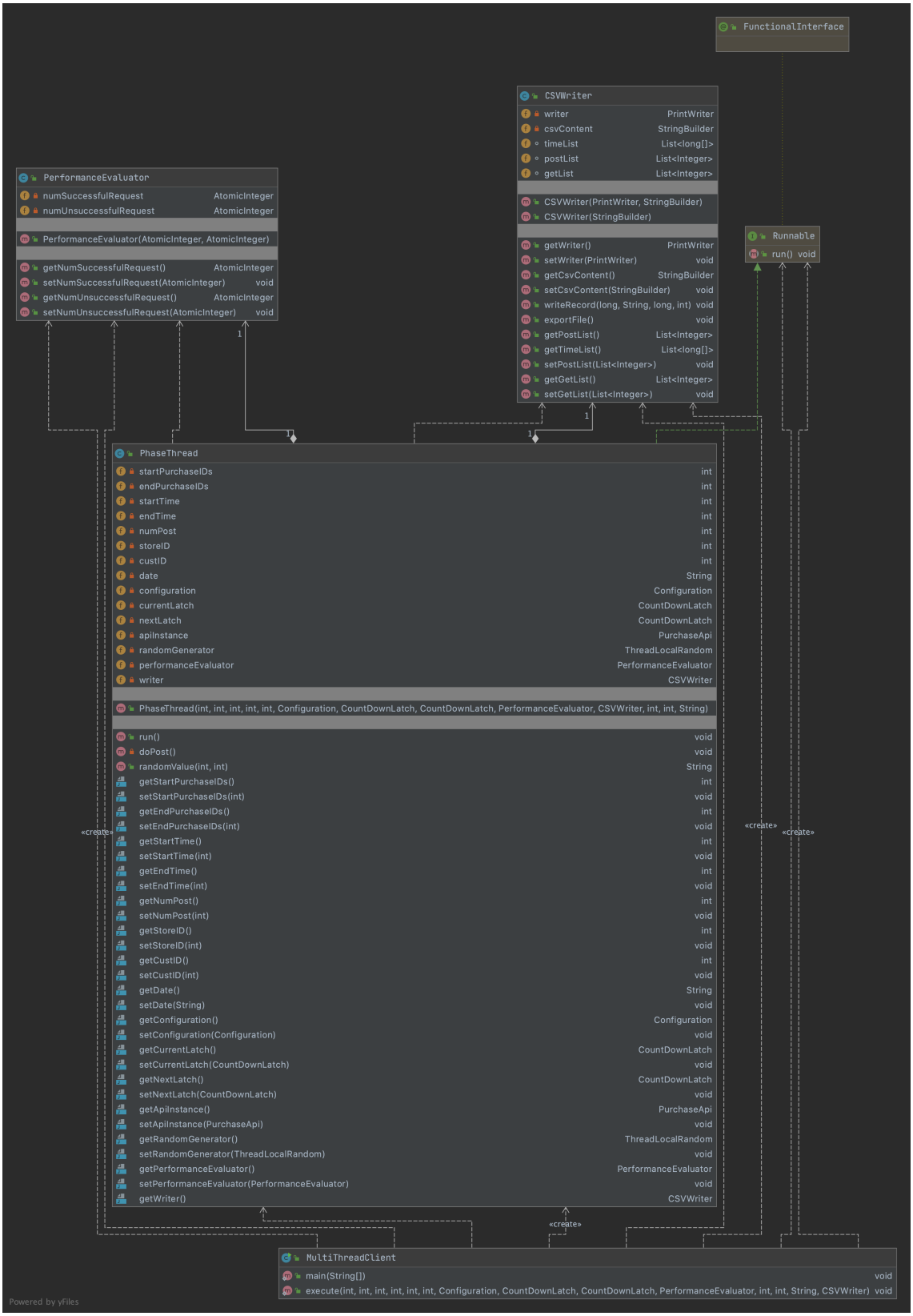
PhaseThread — The class contains the constructor of each single thread. This class would need information like number of purchases, csv writers, store ID, customer ID and performance evaluator to generate the thread (request). Each thread would have a function called doPost() to execute the POST requests. At the same time, the performance evaluator would record the timestamp of each POST request for csv writing.

CsvWriter — The class mainly focus on how to write the csv file. It has a list called timeList to record the start time and end time for each POST request. After all threads executed, the CsvWriter would use the timeList to generate the csv file.

PerformanceEvaluator— The PerformanceEvaluator would track the number of successful request and unsuccessful request.

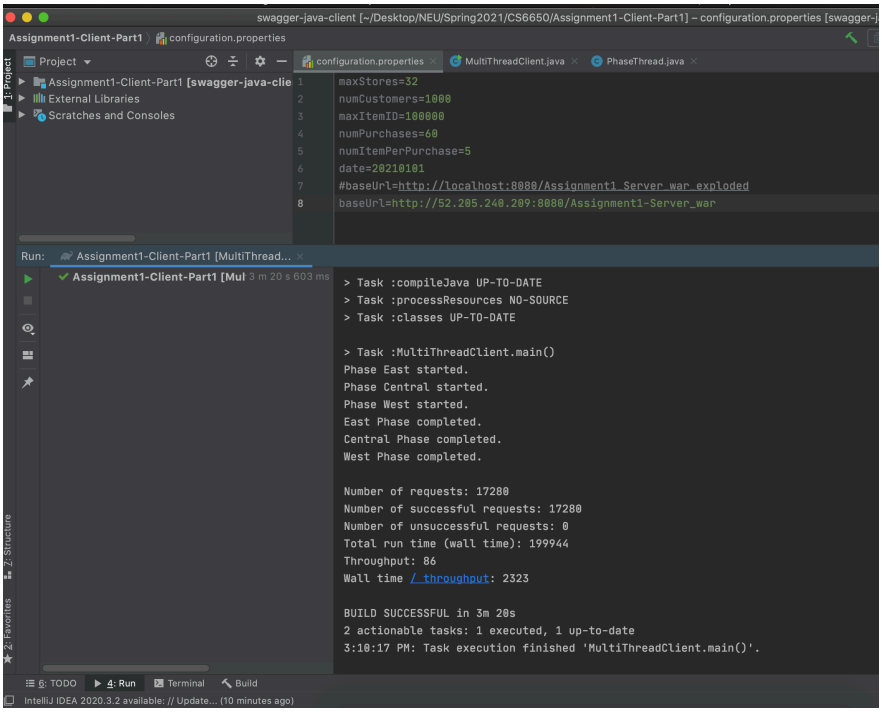
Configuration.properties— A .properties file to store default information such as baseUrl, maxStores, numPurchases and so on. The configuration would be used to transfer datas to the main class to set parameters needed.

UML Diagram:

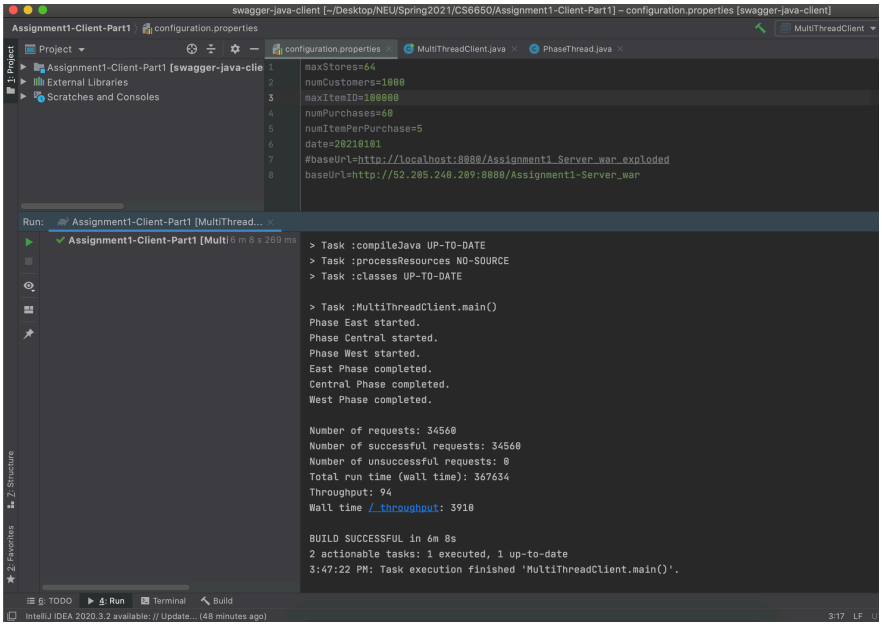

FunctionalInterface
CSVWriter
writer PrintWriter
csvContent StringBuilder
timeList List<long[]>
postList List<Integer>
getList List<Integer>
CSVWriter(PrintWriter, StringBuilder)
CSVWriter(StringBuilder)
getWriter() PrintWriter
setWriter(PrintWriter) void
getCsvContent() StringBuilder
setCsvContent(StringBuilder) void
writeRecord(long, String, long, int) void
exportFile() void
getPostList() List<Integer>
getTimeList() List<long[]>
setPostList(List<Integer>) void
getGetList() List<Integer>
setGetList(List<Integer>) void
PerformanceEvaluator
numSuccessfulRequest AtomicInteger
numUnsuccessfulRequest AtomicInteger
PerformanceEvaluator(AtomicInteger, AtomicInteger)
getNumSuccessfulRequest() AtomicInteger
setNumSuccessfulRequest(AtomicInteger) void
getNumUnsuccessfulRequest() AtomicInteger
setNumUnsuccessfulRequest(AtomicInteger) void
Runnable
run() void
PhaseThread
startPurchaseIDs int
endPurchaseIDs int
startTime int
endTime int
numPost int
storeID int
custID int
date String
configuration Configuration
currentLatch CountDownLatch
nextLatch CountDownLatch
apiInstance PurchaseApi
randomGenerator ThreadLocalRandom
performanceEvaluator PerformanceEvaluator
writer CSVWriter
PhaseThread(int, int, int, int, int, Configuration, CountDownLatch, CountDownLatch, PerformanceEvaluator, CSVWriter, int, int, String)
run() void
doPost() void
randomValue(int, int) String
getStartPurchaseIDs() int
setStartPurchaseIDs(int) void
getEndPurchaseIDs() int
setEndPurchaseIDs(int) void
getStartTime() int
setStartTime(int) void
getEndTime() int
setEndTime(int) void
getNumPost() int
setNumPost(int) void
getStoreID() int
setStoreID(int) void
getCustID() int
setCustID(int) void
getDate() String
setDate(String) void
getConfiguration() Configuration
setConfiguration(Configuration) void
getCurrentLatch() CountDownLatch
setCurrentLatch(CountDownLatch) void
getNextLatch() CountDownLatch
setNextLatch(CountDownLatch) void
getApiInstance() PurchaseApi
setApiInstance(PurchaseApi) void
getRandomGenerator() ThreadLocalRandom
setRandomGenerator(ThreadLocalRandom) void
getPerformanceEvaluator() PerformanceEvaluator
setPerformanceEvaluator(PerformanceEvaluator) void
getWriter() CSVWriter
MultiThreadClient
main(String[]) void
execute(int, int, int, int, int, int, Configuration, CountDownLatch, CountDownLatch, PerformanceEvaluator, int, int, String, CSVWriter) void
«create»
«create»
«create»
«create»
1
1
1
1
Powered by yFiles

Client Part 1

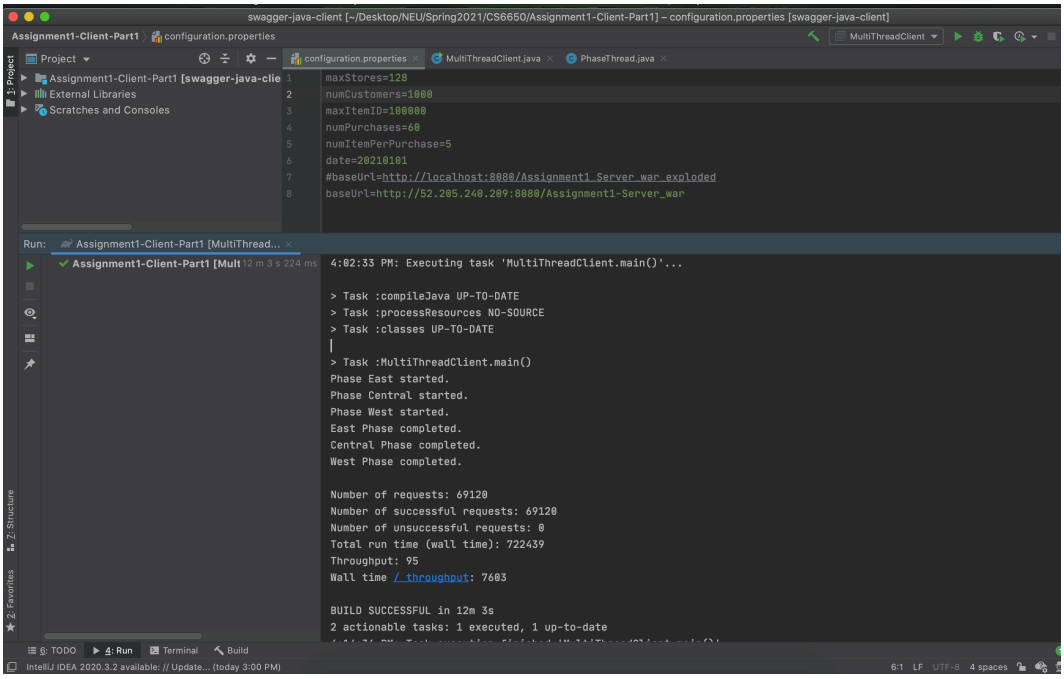
32 Threads:



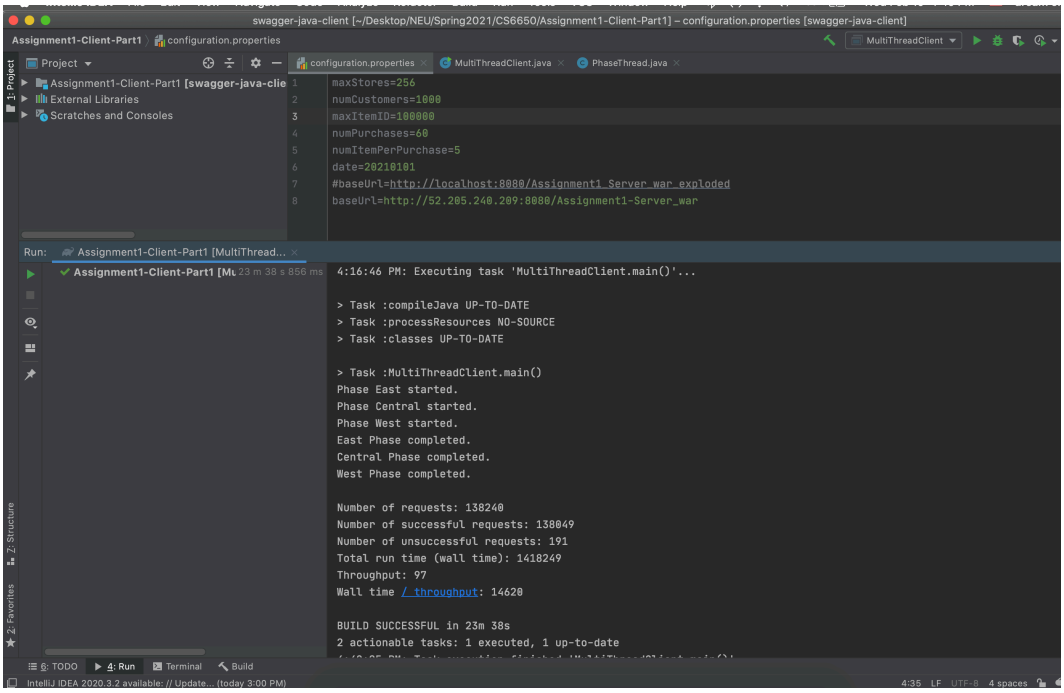
64 Threads:



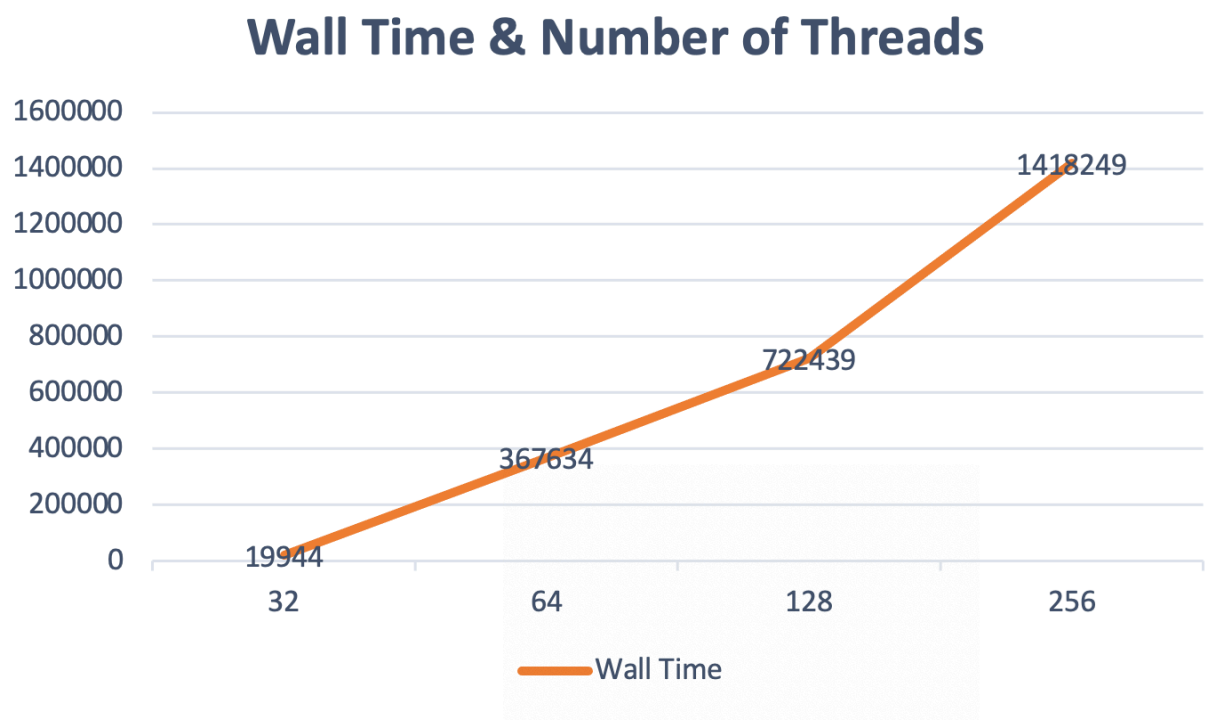
128 Threads:



256 Threads:

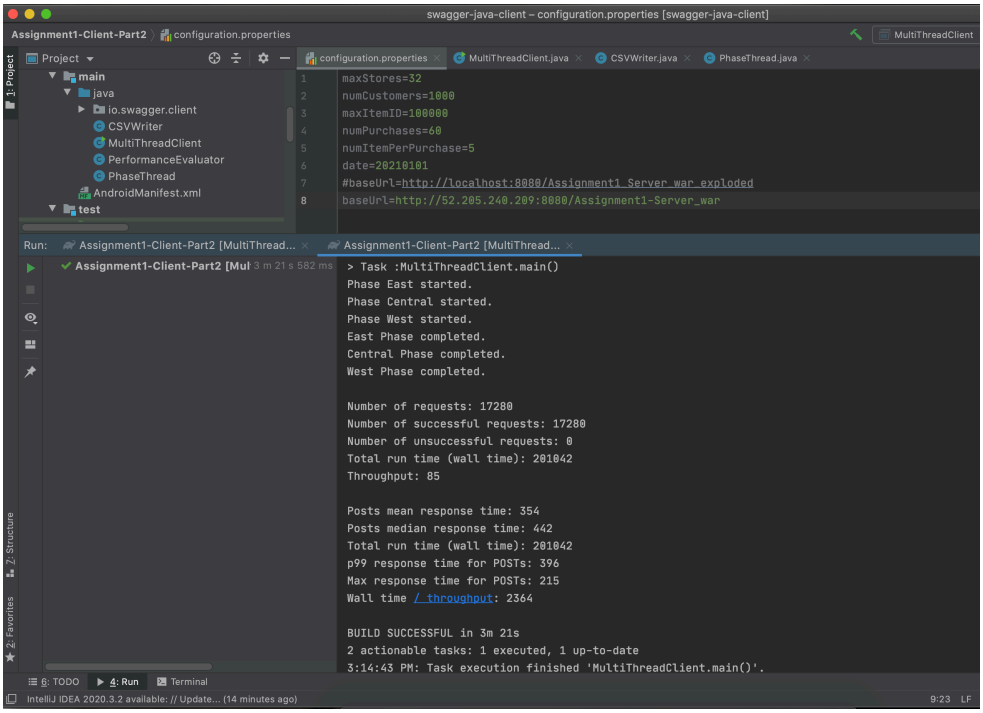


Wall time & number of treads chart:

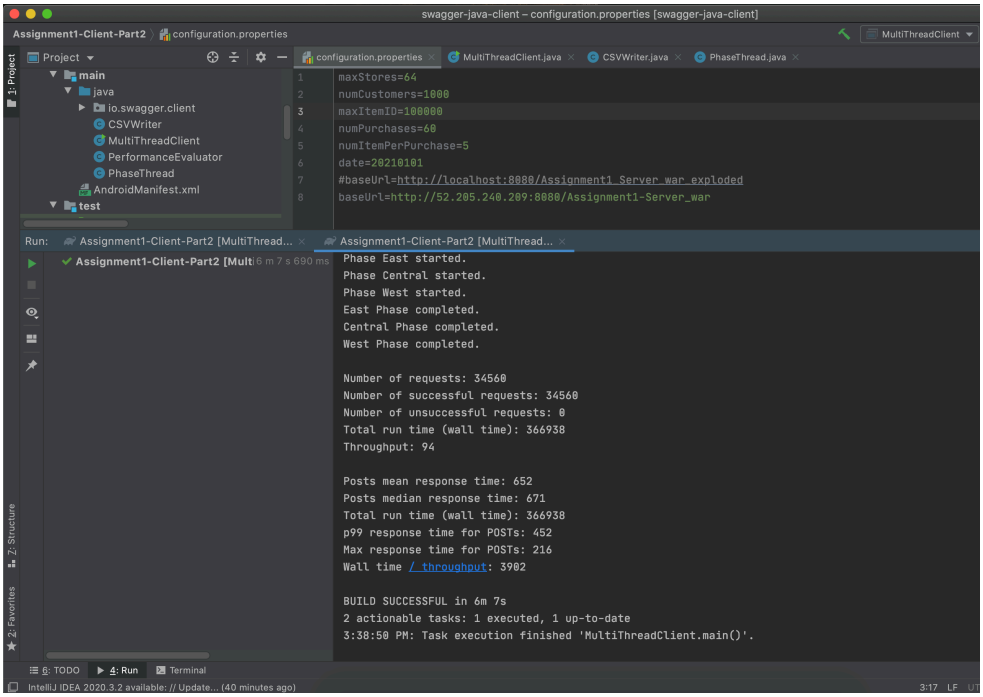


Client Part 2

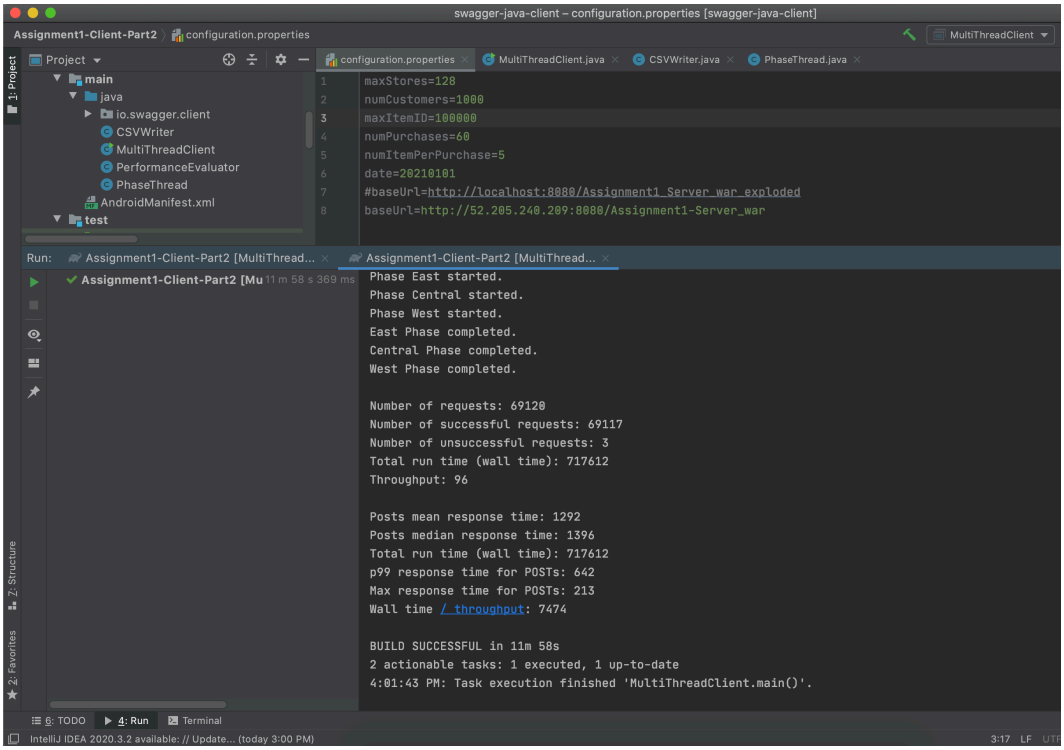
32 Threads (within 5%) :



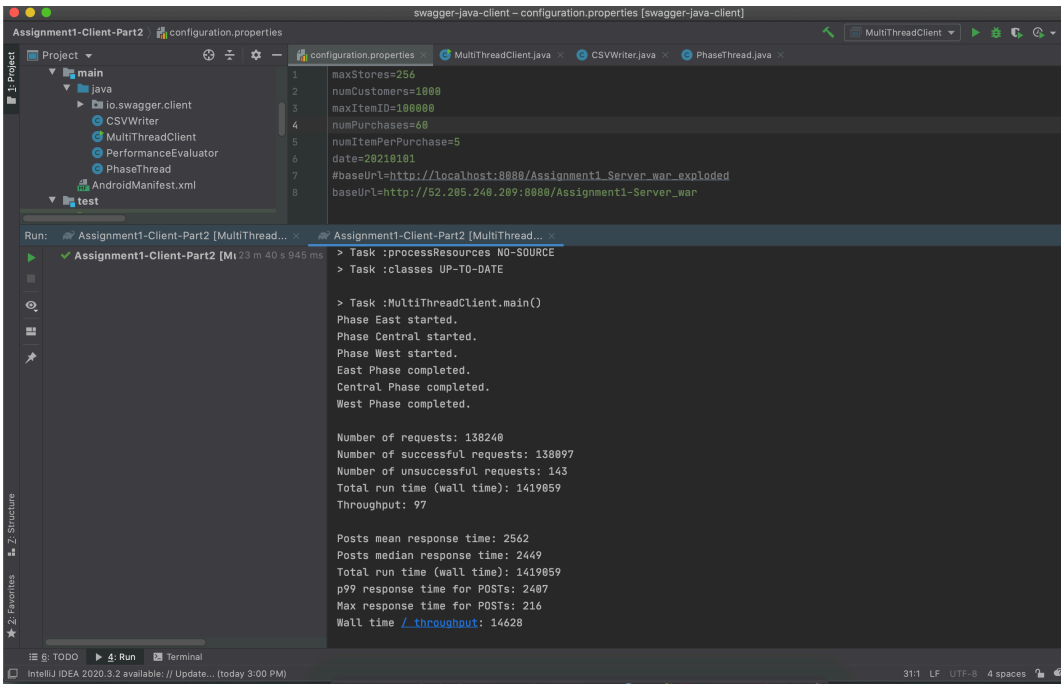
64 Threads (within 5%) :



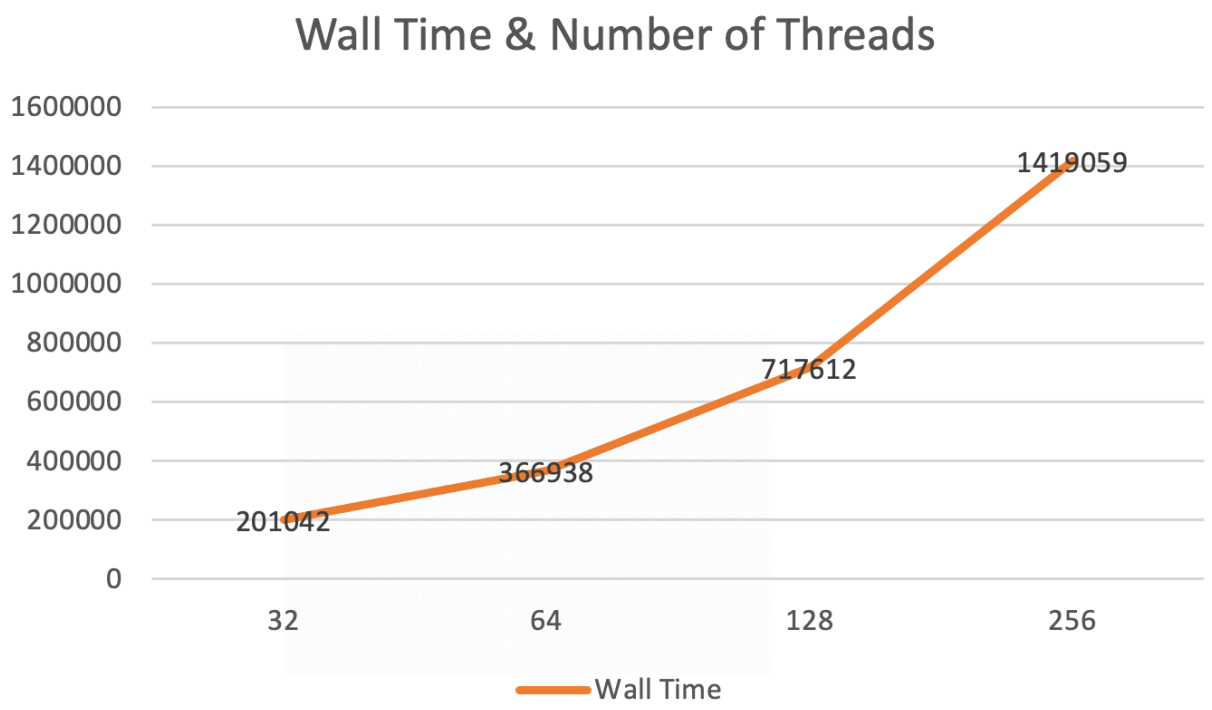
128 Threads (within 5%) :



256 Threads (within 5%) :



Wall time & number of treads chart:



Declarations & Conclusions & Obversions & Guesses:

Here are some declarations, conclusions, obversions and guesses based on the output.

1. For this assignment, I only run a remote sever by using AWS EC2 instances. The client side is still in my local computer. I think that is the reason why my wall time number is very large. Even though the wall time looks large, it makes sense based on my environments set-up and settings. I would run both server and client remotely for next coming assignments. Actually I tried to run the client side remotely but I didn't have enough time to make it. Would keep working on this part.
2. And also because of I didn't run the client remotely, I got some unsuccessful requests when the number of thread is getting large (256). If we look at those dates before we run 256 threads, there is no unsuccessful requests. So it shows the code is working fine and the logic is correct. My guess for this part is the latency caused these unsuccessful requests. Since my client side is run locally, it means the latency could be very "huge" if the thread number getting bigger. And a "huge" latency may cause the server time out for processing the request. Then, unsuccessful requests appeared. Another possible reason to explain these unsuccessful requests could be the EC2 is not stable enough and causes these bad requests accidentally. I hope there is no point lost because of these bad requests.
3. If you look at those charts, you can find that the wall time for each part is close and the difference is within 5%.
4. At first, the difference between two parts is nearly 10% since I tried to write the csv file every request sent. It looks like how to create the csv file smartly is the key point here. What I did was to use a list to record all timestamps and generated the csv file after all phases done.
5. I found those out put dates can be affected by lots of factors such as the speed of your internet, the memory storage of your computer(if you run some codes locally) and the

location of your server. I picked the AWS server at Virginia. So it was kind of slow to run the code.

Break Things:

The client was broken when it reached 1503 threads because of out of memory, based on my situation(only run the server remotely and only 10GB left for my computer).