Client-Part1 design:

For my client class, I focused on creating two classes, a thread runner class `RunningThreads` with methods that will create and execute the threads to make requests to the server, and another class `Client-Part1` that will be used to set the different testing parameters that we are interested in.

In the `RunningThreads` class, I defined two main methods: runThreads and submitTasks. The runThreads function will essentially take in *basePath, threadGroupSize, numThreadGroups, and delaySeconds* arguments, and attempt to create number of thread groups equal to *numThreadGroups*, each group with *threadGroupSize* number of threads, and each thread group, upon finishing execution, will delay for *delaySeconds*. Each thread will then be sending requests to the server defined by the *basePath*.

To achieve creation of threads, I used executor service, and submitted tasks using submitTask. Each task will make use of the `ApiClient` and `DefaultApi` classes from the Java Client SDK from swagger. The submitTask method would make POST using newAlbum and GET using getAlbumByKey. I made a change to the newAlbumCall method in the Java Client SDK because when it sends the POST request, the request body would send the AlbumInfo object instead of its serialized form, so I added the following lines:

```
Gson gson = new Gson();
String profileJson = gson.toJson(profile);
localVarFormParams.put("profile", profileJson);
```

to serialize the AlbumInfo object into json such that the server side logic would not have to case for if the 'profile' data of the request body isn't json.

I have also made use of atomic integers success and failures in the main thread. These would keep track of the number of success requests and failed requests from the api calls in our thread execution. Since we have multiple threads executing, using atomic variables allows safe updates for these global state variables. At the end I would calculate throughput by computing the wall time – which is done by taking the startTime, the time we started thread group executions, and subtract it by endTime, the time we finished all thread group executions – and then take the number of successful requests from atomic integer success and divide it by the wall time.

In order to test, in my ClientPart1class I have defined basePaths for Java, Go servers. If one wants to test Java server, they can just uncomment the tests for the Test Java, and they can do the same for Go.

Client-Part1 design:

Client Part2's main logic remains similar as part1, but it involves keeping track of data involving start Time, method, latency, and request status of each individual POST, GET request from every thread execution. I initially would write to a csv file per request, but then I realized it would be too inefficient since to ensure thread safety we would also have to synchronize the method, and if this were the case then the wait time for all the threads to update and write would slow down the program drastically. So to address this, I used batches. Basically, everytime a request is made, I would store the data temporarily in a batch list, until it reaches capacity of batch size, it would then make the write to csv file. This significantly

reduces the frequency of calling a synchronized write operation. To ensure thread safety, I also made sure the batch is a synchronizedList for thread to update to it.

The last important update involves defining a CSVParser class, which is used to parse the csv data into lists after the threads finish executing and updating the csv. These parsed data will be used to calculate the important metrics. For the step6 calculation, I also specifically defined a step6Calculation method which creates array with size of wallTime, where array[i] = ith second, then it would Go through each data in csv and add the request to figure out the time updated for thread using the request data's startTime to subtract the global startTime (the time we start all the thread tasks) to figure out which second this task belongs to, update array[i], then at the end use plot.py to plot.