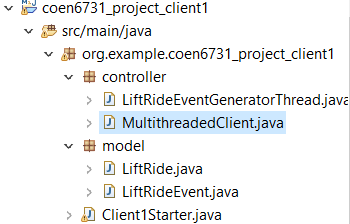
“a 1-2 page report of your client design. Include major classes, packages, relationships, whatever you need to convey concisely how your client works. Include Little's Law throughput predictions. The report should following the IEEE conference paper template”

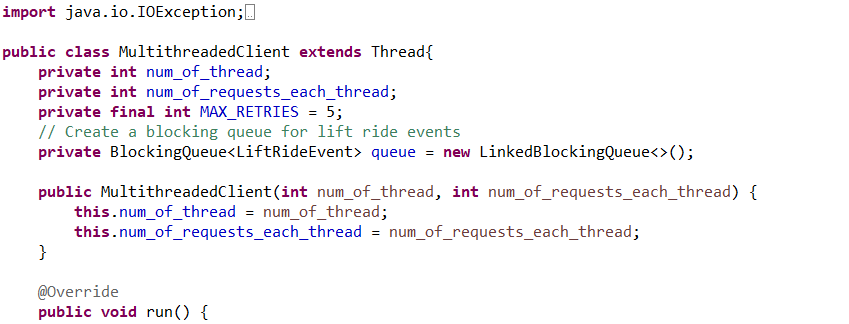
Classes, packages, relationships



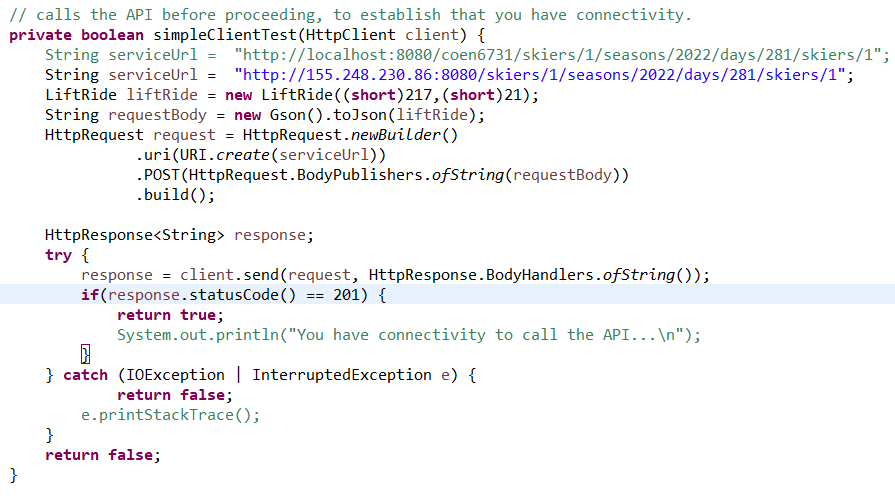
1. In controller package, LiftRideEventGeneratorThread.java is a thread that can be runned to generate the lift ride event data. MultithreaedClient.java is a thread designed to run the HttpClient in multi-threads to send multiple post request to the server API
2. In model package, the LiftRide.java and the LiftRideEvent.java are POJOs.
3. Client1Starter.java is to run the multithreadedClient we built.

Client Design

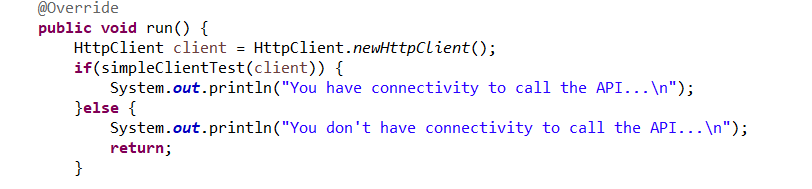
1. Create a Thread to simulate the multithreaded client which can configure the number of threads and number of requests of each thread.



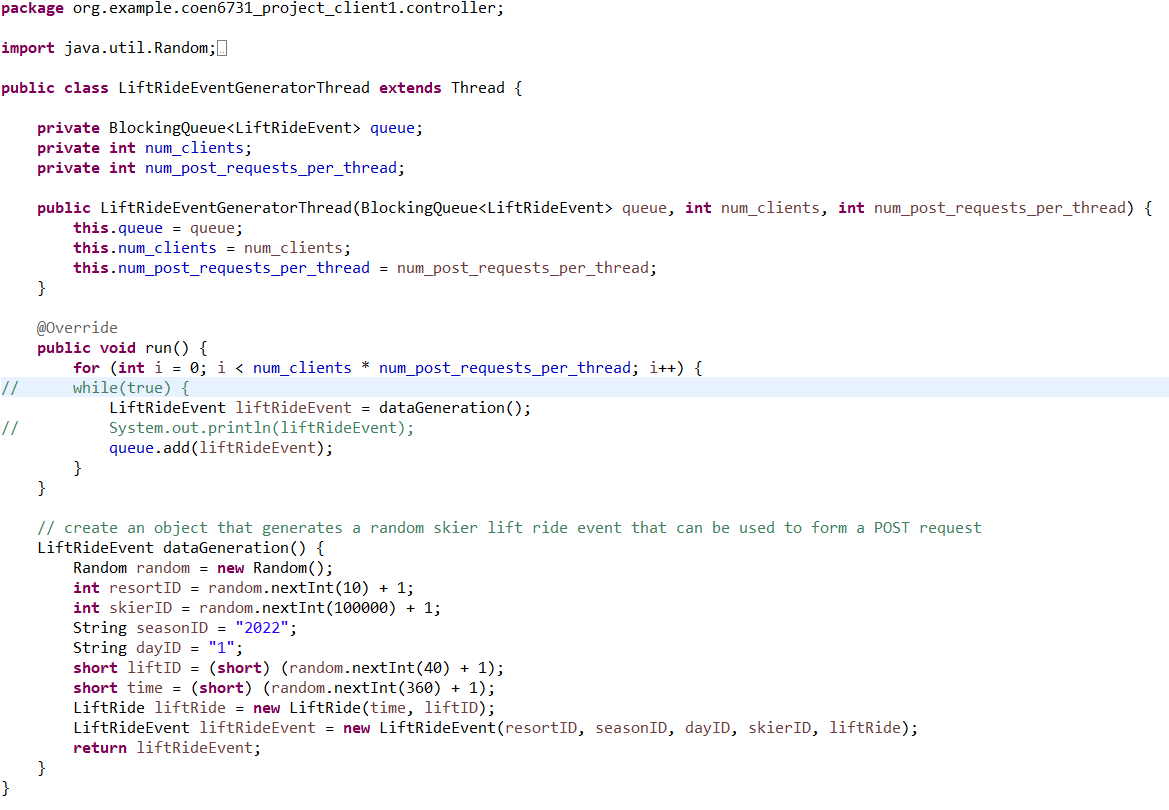
1. Before running the multi threads of the client, write a method to test whether the client has connectivity to the server.



If the client has no connectivity, stop the client thread.



1. Create a data generation thread to generate lift ride event data and store it in the Blockingqueue to keep in the thread-safe memory.



And start the LiftRideEventGeneratorThread in the multithreaded client run() method



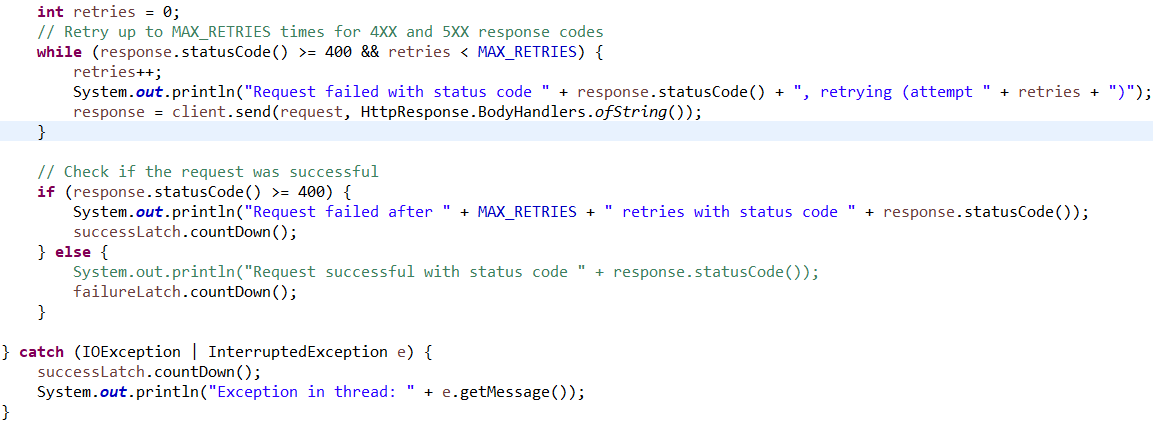
1. Run multiple threads using ExecutorService submit() method. Each thread sends multiple requests. Each request takes the lift ride event object from the blocking queue which stores the lift ride events created by LiftRideEventGeneratorThread before and sends it to the server API.



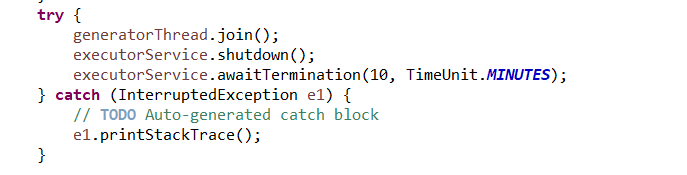
Record the start time before running multiple threads, and store each response time in a thread-safe list.



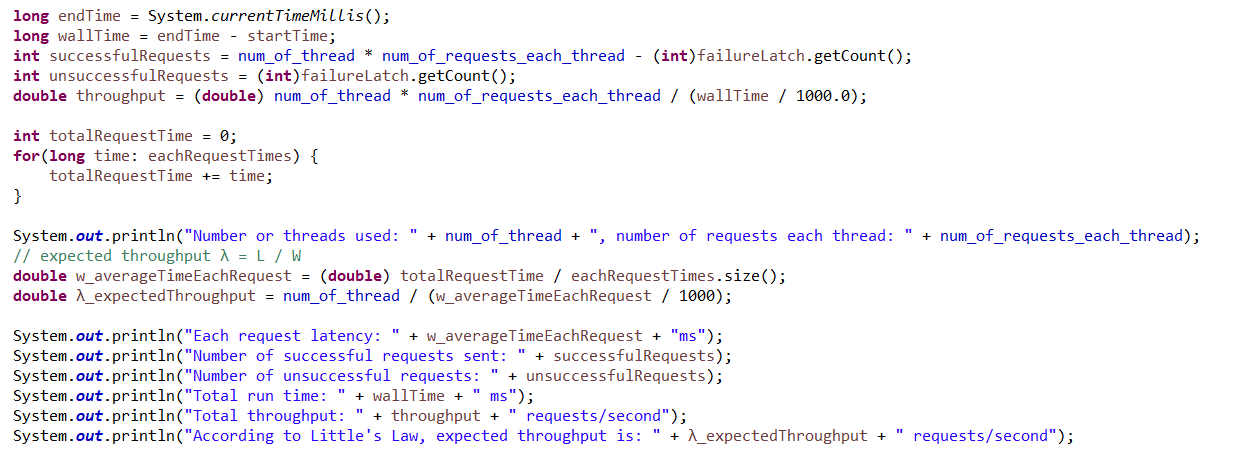
If the client receives a 5XX response code (Web server error), or a 4XX response code (from your servlet), it should retry the request up to 5 times before counting it as a failed request.



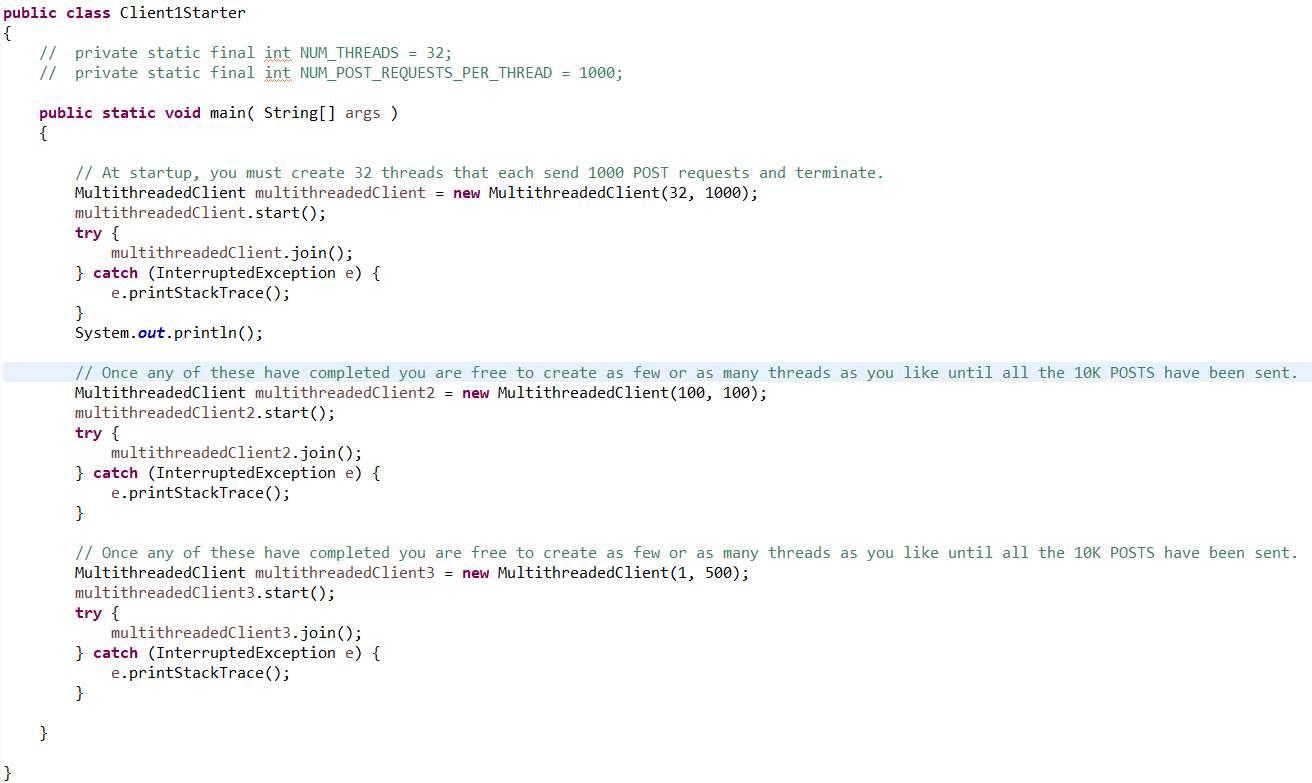
When all 10k requests have been successfully sent, all threads should terminate cleanly.

The programs should finally print out:

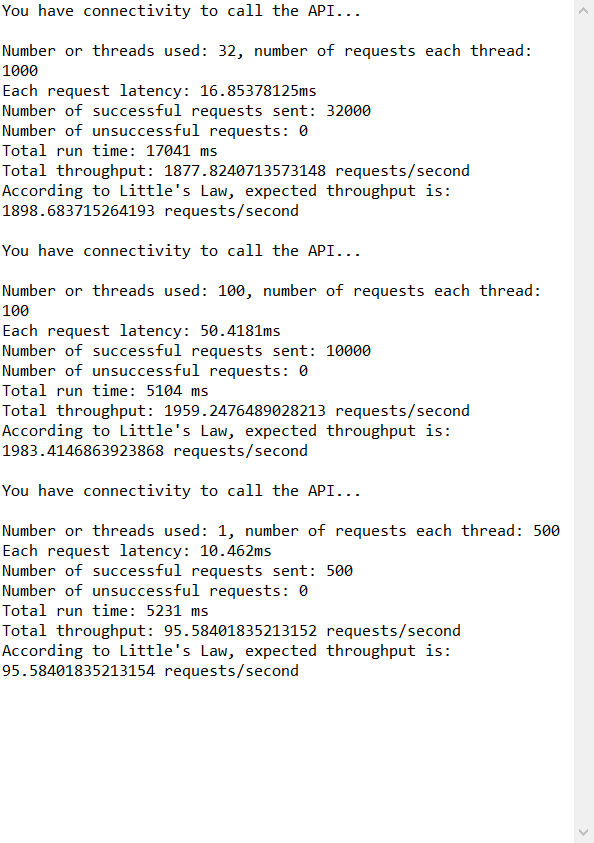
1. number of successful requests sent
2. number of unsuccessful requests (should be 0)
3. the total run time (wall time) for all phases to complete. Calculate this by taking a timestamp before you start any threads and another after all threads are complete.
4. the total throughput in requests per second (total number of requests/wall time)
5. The predicted throughput according to Little’s Law



5. In main entry Client1Starter.java, we create run the MultithreadedClient thread in different number of threads and number of requests per thread.



The output window:



The real throughput is closed to the predicted throughput, client building successfully.