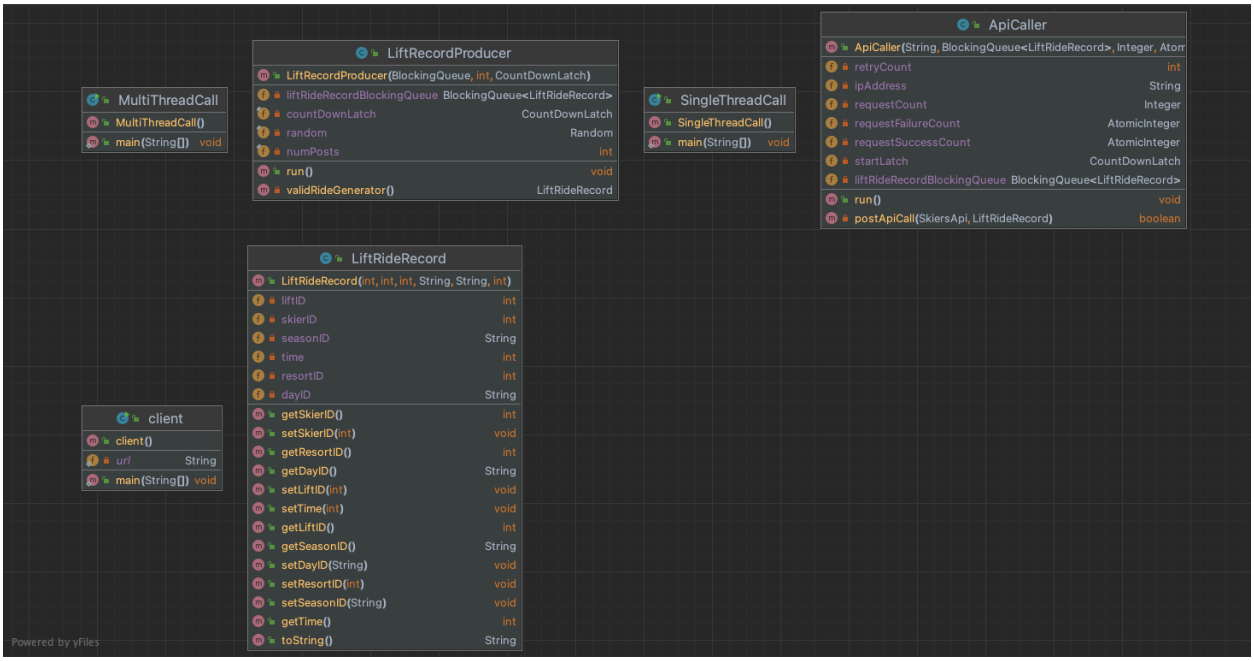
Assignment 1 Zegui Jiang

Client 1



1 - The ApiCaller class is designed to manage API calls to a ski Server, encapsulating the logic for posting lift ride data. Implementing the Runnable interface, this class is intended for use in concurrent execution environments, allowing it to be run on separate threads to facilitate parallel API requests.

run () method: Overriding the Runnable interface's run method, it contains the logic to sequentially take LiftRideRecord objects from the queue and post them to the ski resort service API. It utilizes a retry mechanism for handling API call failures. Once the designated number of requests is processed, it updates the success and failure counts and decrements the start latch, signaling completion.

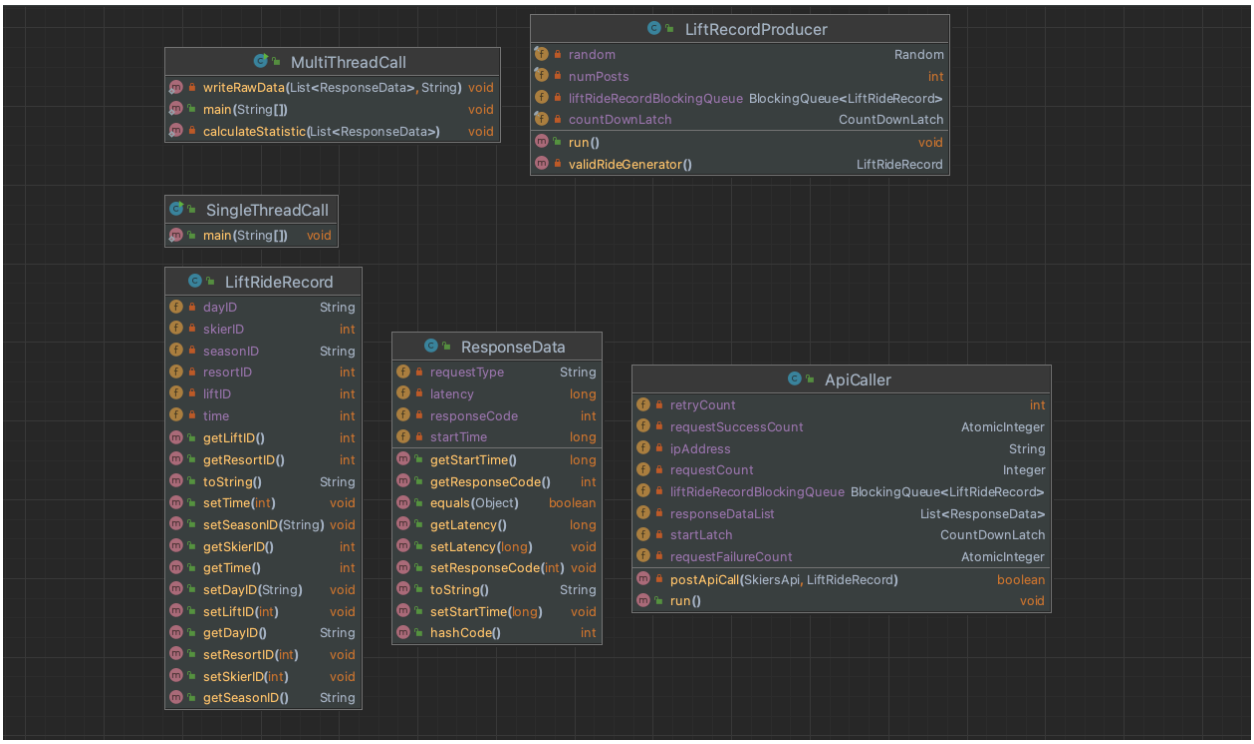
postApiCall method: A private helper method that executes the actual API call to post a LiftRide object. It implements a simple retry strategy based on the predefined retryCount for handling request failures due to network issues or server errors.

2 - The LiftRideRecord class, contained within the model package, serves as a data model representing a record of a ski lift ride within a skiing resort application.

3 – LiftRecordProducer simulate the production of ski lift ride records and enqueue them into a BlockingQueue for further processing. This class implements the Runnable interface, allowing it to be executed by a thread.

4 - MultiThreadCall class is designed to simulate a multi-threaded environment for making API calls to a server, specifically targeting the scenario of logging ski lift rides at a ski resort. This class efficiently manage and execute multiple threads for API calls and data production.

Client 2



Client 2 extends the functionality of client 1 by enhancing the MultiThreadCall class to include capabilities for writing raw data to a CSV file and calculating statistical metrics. To fulfill these additional requirements, a new class named ResponseData is introduced to encapsulate the response records.

The MultiThreadCall class in client 2 would have these new responsibilities:

Writing Response Records: After each API call, the response details are captured in an instance of the ResponseData class. This object includes all relevant information that needs to be logged, such as timestamps, response status, and any payload data returned by the API.

Calculating Statistics: The class is responsible for computing various performance metrics based on the response data collected. These statistics might include the total number of requests, success rate, failure rate, average response time, and other relevant performance indicators.

Littles law:

Base on little's law, $N = \lambda W$.
if we predict 100 threads. Throughput = 100 / 35 * 1000 = 2857
the actual performance is 2524 is close to 2857.

Performance between client 1 and client 2

```
Summary:
Number of thread in process 1: 32
Number of thread in process 2: 100
Number of successful requests: 200000
Number of fail requests: 0
Total run time: 98848
Response Time: 0.49424 ms/request
RPS: 2023 requests/second
```

```
/Users/monarch/Library/Java/JavaVirtualMachines/corretto-11.0.22/Contents/Home/bin/java ..
Statistic Metrics
Mean Response Time: 39.634475 ms
Median Response Time: 33.0 ms
P99 Response Time: 128 ms
Min Response Time: 15 ms
Max Response Time: 272 ms
Summary:
Number of thread in process 2: 100
Number of successful requests: 200000
Number of fail requests: 0
Total run time: 103922
Response Time: 0.51961 ms/request
RPS: 1924 requests/second

Process finished with exit code 0
```

We can see the difference for throughput within 5% of Client Part 1

Optimization: Throughput Overtime

```
Statistic Metrics
Mean Response Time: 38.736505 ms
Median Response Time: 32.0 ms
P99 Response Time: 125 ms
Min Response Time: 14 ms
Max Response Time: 594 ms
Summary:
Number of thread in process 2: 50
Number of successful requests: 200000
Number of fail requests: 0
Total run time: 167183
Response Time: 0.835915 ms/request
RPS: 1196 requests/second
```

```
/Users/monarch/Library/Java/JavaVirtualMachines/corretto-11.0.22/Contents/Home/bin/java ..
Statistic Metrics
Mean Response Time: 39.634475 ms
Median Response Time: 33.0 ms
P99 Response Time: 128 ms
Min Response Time: 15 ms
Max Response Time: 272 ms
Summary:
Number of thread in process 2: 100
Number of successful requests: 200000
Number of fail requests: 0
Total run time: 103922
Response Time: 0.51961 ms/request
RPS: 1924 requests/second

Process finished with exit code 0
```

```
/Users/monarch/Library/Java/JavaVirtualMachines/corretto-11.0.22/Contents/Home/bin/java ...
Statistic Metrics
Mean Response Time: 38.10257 ms
Median Response Time: 31.0 ms
P99 Response Time: 125 ms
Min Response Time: 16 ms
Max Response Time: 429 ms
Summary:
Number of thread in process 2: 200
Number of successful requests: 200000
Number of fail requests: 0
Total run time: 67915
Response Time: 0.339575 ms/request
RPS: 2944 requests/second

Process finished with exit code 0
```

```
Statistic Metrics
Mean Response Time: 47.327139980110644 ms
Median Response Time: 37.0 ms
P99 Response Time: 155 ms
Min Response Time: 15 ms
Max Response Time: 2067 ms
Summary:
Number of thread in process 2: 400
Number of successful requests: 200000
Number of fail requests: 0
Total run time: 55637
Response Time: 0.278185 ms/request
RPS: 3594 requests/second
```

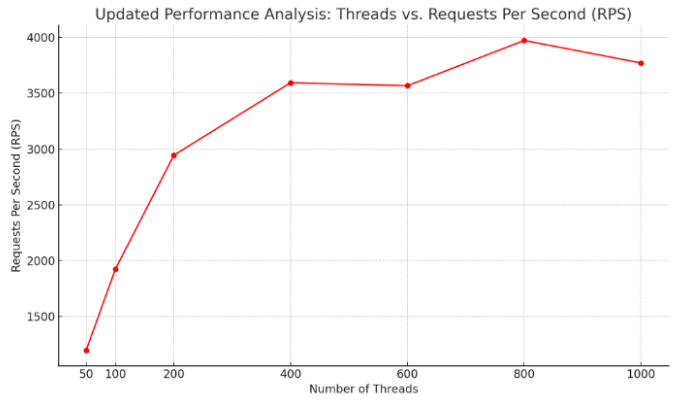
```
Statistic Metrics
Mean Response Time: 69.36437645553679 ms
Median Response Time: 57.0 ms
P99 Response Time: 302 ms
Min Response Time: 13 ms
Max Response Time: 1947 ms
Summary:
Number of thread in process 2: 600
Number of successful requests: 200000
Number of fail requests: 0
Total run time: 56060
Response Time: 0.2803 ms/request
RPS: 3567 requests/second
```

```
Statistic Metrics
Mean Response Time: 82.46096060097499 ms
Median Response Time: 65.0 ms
P99 Response Time: 361 ms
Min Response Time: 17 ms
Max Response Time: 10288 ms
Summary:
Number of thread in process 2: 800
Number of successful requests: 200000
Number of fail requests: 0
Total run time: 50334
Response Time: 0.25167 ms/request
RPS: 3973 requests/second
```

```
Mean Response Time: 95.08575497863622 ms
Median Response Time: 74.0 ms
P99 Response Time: 439 ms
Min Response Time: 16 ms
Max Response Time: 10219 ms
Summary:
Number of thread in process 2: 1000
Number of successful requests: 200000
Number of fail requests: 0
Total run time: 53023
Response Time: 0.265115 ms/request
RPS: 3771 requests/second
```

Summary:

Based on the requirements, Process 1 will be configured with a fixed number of 32 threads, with each thread handling 1000 requests. Once any thread completes its tasks, we will initiate Process 2, which operates with an optimized number of threads and a corresponding number of requests per thread.



The graph illustrates the performance analysis based on the number of threads versus the Requests Per Second (RPS). It shows a general upward trend in RPS as the number of threads increases. The performance increases significantly up to 800 threads, marking the peak RPS achieved in the data set. However, increasing the thread count further to 1000 results in a slight decrease in RPS, indicating a potential point of diminishing returns where adding more threads no longer contributes to performance improvements and may even reduce efficiency.

Thanks Zegui