ASSIGNMENT 1

(CS 6650 - Building Scalable Distributed Systems)

Execution Results:

Running logs:
https://github.com/rahulpandeycs/bsds6650-Course-fall2020/tree/master/Assignment%201/bsds-cs6650-hw1-clientPart1/Run_Screenshots

https://github.com/rahulpandeycs/bsds6650-Course-fall2020/tree/master/Assignment%201/bsds-cs6650-hw1-clientPart2/run_Screenshots

Plotting Data:

https://github.com/rahulpandeycs/bsds6650-Course-fall2020/tree/master/Assignment%201/bsds-cs6650-hw1-clientPart1/PlotCsv

https://github.com/rahulpandeycs/bsds6650-Course-fall2020/tree/master/Assignment%201/bsds-cs6650-hw1-clientPart2/PlotCsv

Generated CSV:
https://github.com/rahulpandeycs/bsds6650-Course-fall2020/tree/master/Assignment%201/bsds-cs6650-hw1-clientPart2/csv_output

UML diagrams:

https://github.com/rahulpandeycs/bsds6650-Course-fall2020/tree/master/Assignment%201

Executable Jars:
https://github.com/rahulpandeycs/bsds6650-Course-fall2020/tree/master/Assignment%201/bsds-cs6650-hw1-clientPart1/executable_jar

https://github.com/rahulpandeycs/bsds6650-Course-fall2020/tree/master/Assignment%201/bsds-cs6650-hw1-clientPart2/executable_jar

Running the application:

The application is divided into 3 Parts:

- The Server
- The Client Part 1
- The Client Part 2

The server needs to be hosted and kept running either on Cloud (e.g AWS) or Localhost. The client will then need to modify the **resources/config.properties** to point to its address and execute calls.

A sample view of contents of config.properties looks like:

1. maximum number of threads to run (maxThreads - max 256)
2. number of skier to generate lift rides for (numSkiers - default 50000), This is effectively the skier's ID (skierID)
3. number of ski lifts (numLifts - range 5-60, default 40)
4. the ski day number - default to 1
5. the resort name which is the resortID - default to "SilverMt"
6. IP/port address of the server

Config.properties

cmd.maxThreads=32
cmd.numSkiers=50000
cmd.numLifts=60
cmd.skiDay=1
cmd.resortId=SilverMt
#Local
cmd.addressPort=http://localhost:8081/CS6650Assignment1Server_war_exploded

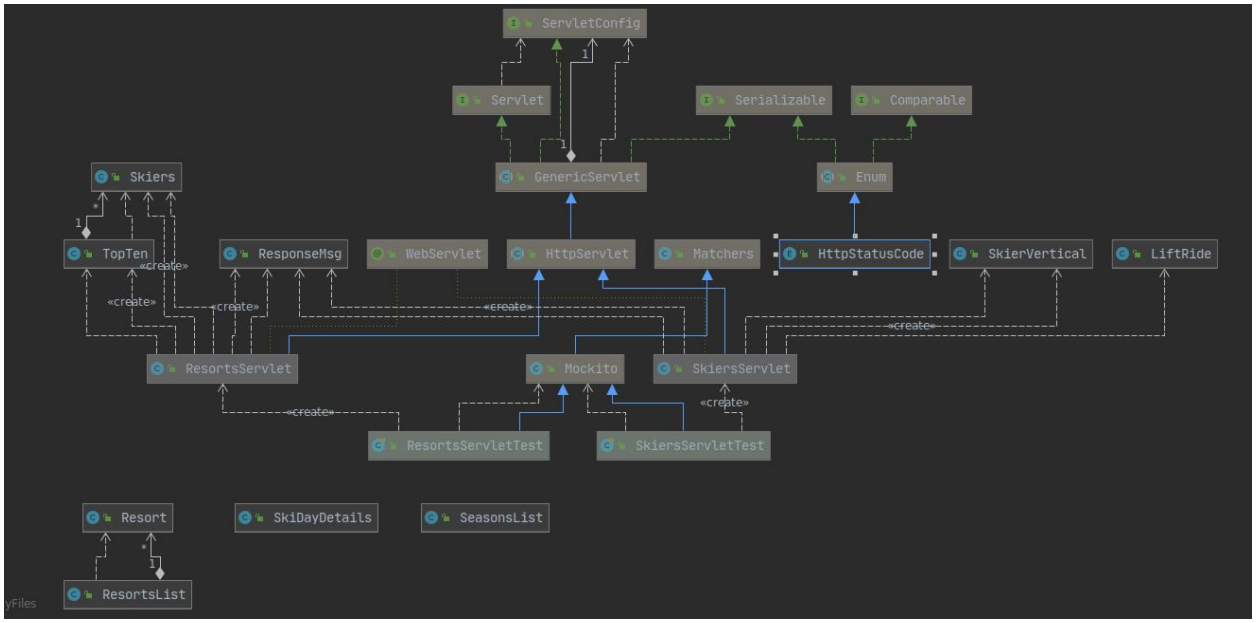
To run the application, the client needs to be packaged as .jar with configured config.properties. Then run as below:

Client part1: (Jar included in folder executable_jar)
java -jar commandLine_run_hw1-clientPart1.jar -f config.properties

Client Part2: (Jar included in folder executable_jar)
java -jar commandLine_run_hw1-clientPart2.jar -f config.properties

Note: If no config.properties is provided it reads default config.properties

The Server



The server exposes below API using Java Servlets:

resorts

GET/resort/day/top10vert

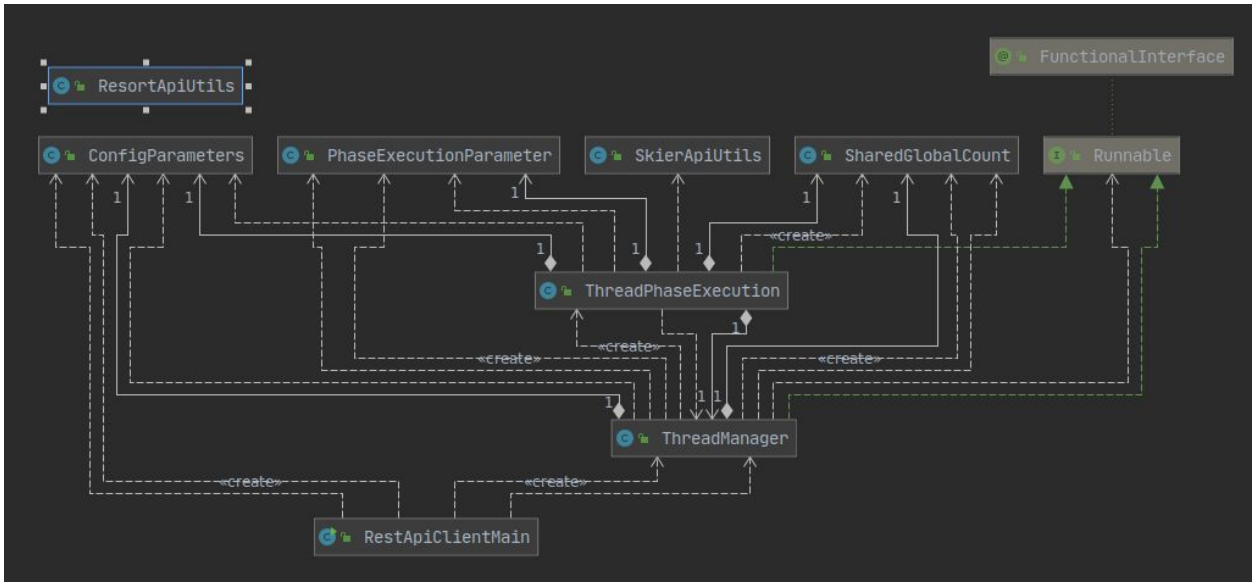
skiers

POST/skiers/liftrides

GET/skiers/{resortID}/days/{dayID}/skiers/{skierID}

GET/skiers/{skierID}/vertical

The Client Part 1



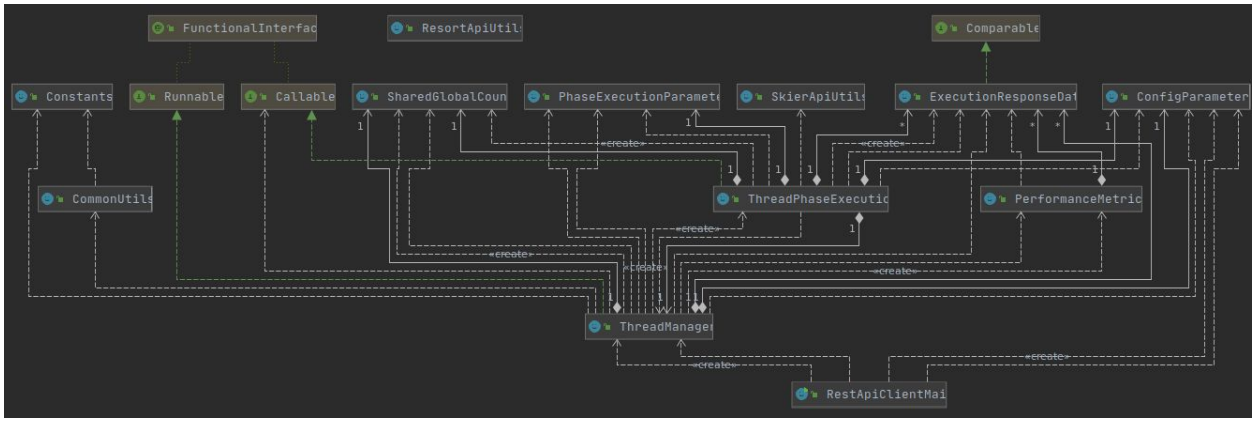
The Client has been designed as a multi threaded client. The client execution starts with execution of the jar file and clients main class *RestApiClientMain* is called. The main class parses the input config.properties into a properties file and initializes *ConfigParameters* class. The control is then passed to *ThreadManager* class which is executed in a separate thread then main. The threadManager takes care of threads phase execution by calling *ThreadPhaseExectution* class with their respective configuration as initialized in *PhaseExecutionParameter* Class.

The respective phases are then executed and after 10% of each phase is elapsed the next phase is started,this is taken care of by using CountDownLatch and initializing it to 10% of the total number of threads for a respective previous phase.

The execution then terminates with results printed to the console. A sample output will look like:

```
Number of successful requests sent : 5080
Number of unsuccessful requests :0
The total run time (wall time) :4381 ms
Throughput: 1159 Requests/Second
```

The Client Part 2



The Client has been designed as a multi threaded client. The client execution starts with execution of the jar file and clients main class *RestApiClientMain* is called. The main class parses the input config.properties into a properties file and initializes *ConfigParameters* class. The control is then passed to *ThreadManager* class which is executed in a separate thread then main. The threadManager takes care of threads phase execution by calling *ThreadPhaseExectution* class with their respective configuration as initialized in *PhaseExecutionParameter* Class.

The respective phases are then executed and after 10% of each phase is elapsed the next phase is started,this is taken care of by using CountDownLatch and initializing it to 10% of the total number of threads for a respective previous phase. As results need to be returned for each executed thread, Callable is used instead of runnable as in client part 1. The results are stored in list of *ExecutionResponseData* class, this list is then passed to *PerformanceMetrics* class which takes care of generating mean, median and various other performance metrics.

The execution then terminates with results printed to the console and CSV is generated from run results as stored in List of *ExecutionResponseData* Class. A sample output will look like:

```
[ec2-user@ip-172-31-89-98 bsds_assignment1]$ java -jar 32Threads_bsds-cs6650-hw1-clientPart2.jar
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Mean responseTime:42 ms
Median responseTime:36 ms
The total run time (wall time) :4778 ms
Total Requests: 5080
Throughput: 1063 requests/sec
p99 (99th percentile) response time :172 ms
Max response time:592 ms
Writing data to CSV
Successfully Completed, Results are stored at: /usr/bsds_assignment1/performance_metrics.csv
```