CLIENT1

- Package Name - edu.neu.ds.load
- Classes
  - Client1.java
    - This file reads the input from the text file (client_input.txt)
    - The input is read line by line and the execution is done for 4 inputs(32, 64,128,256) one after the other synchronously.
    - Sample line in input = maxThreads=32,skiers=20000,skiLifts=40,numRuns=20
    - Note down the currentTime(**StartTime)**
    - Considering above sample input, First ExecutorService(for Phase1) is created with (maxThreads/4) threads and is given Task1(described below) to execute
    - Once 10% of threads complete in Phase1, ExecutorService for Phase2 is created with number of threads(maxThreads) and is given Task1.
    - This is similarly done for Phase3
    - Ensure all the Phases are completed.
    - Note down the current time(**End Time**)
    - Calculate the **wallTime** = EndTime - StartTime
    - Shutdown the executor services
    - Calculate the wallTime(for 32 maxThreads) and return the result back to the caller(main method)
    - The execution results of (32 threads) is collected in List
    - Start execution for 64 threads and after it completes for 128 and then 256
    - Using the results in the list for each execution write to CSV
    - How Phase2 is started after 10% rounded up of Phase1 ?
      - Used a CountdownLatch for example, in Phase1 and set it to (Math.ceil of 10% of (32/4))
      - This is passed to the Task1. Task1 will call latch.countDown() to decrement when a thread completes.
      - Client1 will await till countdownLatch decreases the count for 10% of the threads
  - Task1.java - Class
    - This class implements Runnable
    - Has a constructor that takes in
      - Range of skiers
      - Start and end time range
      - 2 instances of CountDownLatch. 1 for 10% threads completion and another for all the threads completion
      - NumRuns
    - In the Run method
      - Randomly select skierId and time and liftId from the given ranges
      - Create the Lift Object
      - Send a HTTP POST request using the SDK
      - Use try-catch on the POST request to identify errors
      - Loop this based on numRuns
      - Execute countDown on Latch once the thread completes the work
  - Client1Data.java - Class

            - Simple class that maintains the output of each line executed in the input file

CLIENT2

- Package Name - edu.neu.ds.analyze
- Classes
    - Client2.java
        - This is similar to Client1 with respect to reading the input data from the file.
        - Input file is different file for Client2 but has the same input content as Client1
        - Creation of Executor Service and usage of CountdownLatch is the same
        - Similarly calculate the Wall Time as done in Client1
        - How Number of Passed requests and Number of Failed requests are calculated?
            - Use 2 AtomicInteger objects for each Phase. One for SuccessFul completion of a POST request, Another for Error from the Server
            - Pass these as part of the Task2 class
            - Data calculated for each phase can be aggregated after all phases are done
        - Once all the phases are done, find out WallTime like Client1
        - Calculate throughput = (Aggregated Total Requests from Atomic Integers of all phases) / Wall time
        - This also has a List<Data> object, where Data is a simple class that maintains all the records that need to be written in the final CSV. Data has properties for each request like (StartTime, EndTime, latency, WhichNumRun, ThreadName, WhichPhase)
        - This is passed to the Task2 class
        - Through List<Data> after all the phases complete we get the individual latencies for all requests.
        - Used Apache library to calculate, median, mean, percentile calculation
        - All this logged to the records.csv(Each maxThreads execution 32, 64, 128, 256) have their own records.csv
        - Use class Client2Data to return back MaxresponseTime,ThroughPut to the caller for the current maxThreads execution.
        - This is added to the List by the caller.
        - Once all maxThreads execution are done, this list is written to a CSV
    - Task2.java
        - Has Similar constructor as Task1. Additional parameters are
            - 2 AtomicIntegers for SuccessFulRequest count and Error Count
            - List<Result> object from Client2 class
        - The Run method
            - Will create random skierId, time, liftId like Client1
            - Create lift object
            - Note down StartTime before doing HTTP POST
            - Do the POST
            - Note down EndTime after receiving the success/error response
            - Based on response code increment the appropriate AtomicInteger
            - Now we have the startTime, endTime, latency, responseCode for a single request

- Create Data object from the above props and add it to a local list object of that Run method.
- This will have all the records for the current thread.
- Once the thread completes the task, use the List<Data> object that is passed as constructor arg. This contains the data for all threads
- Since this is a shared object between threads. This object needs to be syncronized before the data for each thread is added