Directions

A. Due Date: 3 February 2025 at 11:55pm

B. The homework may be done in groups of up to two students.

What to turn in

C. For the programming sections, turn in all related source code (.java files) along with any headers or supplemental libraries needed to compile the code.

- Do not turn in compiled JAR files (*i.e.* the entire build/ directory should not be uploaded to the repository).

D. The GitHub repository includes several GitHub Actions scripts that will compile your code and perform test runs on it. You can see the output from the actions scripts in the Actions tab on GitHub. Those outputs will be the basis for your grade.

E. Turn in all related source code along with any headers or build files needed to compile the code in the GitHub repository.

F. In addition, turn in a README.md file with the following:

- Names of all students in the group
- Total number of hours spent on each part of the assignment
- Date of submission
- Comments or feedback on the assignment's requirements or complexity (optional)

G. The README.md and GitHub submission practices are worth 5 points on the assignment.

How to submit

H. Turn in your submission via the starter repositories opened for you on GitHub via GitHub Classroom.

I. All submissions **must** be made in the starter GitHub repository for the assignment that is opened via GitHub classroom. Code placed anywhere else or submitted in any other manner - including via email or in unrelated repositories - will not be graded.

J. Indicate that your work is complete by performing a commit with the comment "Submitted for grading" on each repository. The grade for the assignment will be based on the first commit with that comment, so create a submission with that comment only when you are finished.

Subnet Routing Table, TCP, Threads

In this assignment, we will write a calculator that computes subnet routing decision based on a routing table. The routing table will contain subnet numbers, subnet masks, and next hops. The calculator will take an input address, look it up in the routing table, and output the routing decision.

To make the tool more interesting and to exercise additional networking concepts, we will build the calculator in two parts - a client and router - that will communicate using TCP sockets.

1 Client Tool

The client tool will receive the following parameters from the command line:

dest The IP address of the router. It must be an IPv4 address written in the typical four byte format (*e.g.* 10.10.10.10).

port The port the router listens on. It must be an integer value [1025, 65535). (*e.g.* 5000, 6000).

addresses A comma separated list of addresses to send to the router. The list must not have spaces. Example: 10.0.0.1,10.1.1.0,5.50.1.50

1.1 Illegal parameters and addresses

If any of the parameters are missing or illegal, quit with a usage message.

The exception to that rule is that the client does not parse or attempt to identify illegal addresses in the address list (*e.g.* 21.5.2). The client will send the address to the router. The router will then return an error message ("Illegal address: 21.5.2") which the client will output.

1.2 Client Algorithm

The client performs the following algorithm:

1. Open a TCP connection to the router using the destination IP address and port
2. Send 1 address from the list of addresses to the router. The address is sent on a line (*i.e.* with `println`).
3. Receive the response from the router. The response is sent on a line (*i.e.* with `println`).
4. Output the response if the next hop is found. Example: `10.0.0.1 -> R3, 5.50.1.50 -> R2`
5. If the response is "NotFound", output an appropriate message. Example: 10.1.1.0 not found
6. If there are more addresses in the list, repeat from step 2.
7. When all addresses are finished, close the TCP connection.

The client outputs some logging information about connections.

1.3 Client output formats

To make your programming go a bit faster, I provided String formats for all expected outputs in a class called `ClientConstants.java`. You are welcome to use them to make your outputs conform to the autograder's checks more easily.

1.4 Client Documentation (10 points)

Add Javadoc documentation to every method and class. The documentation for methods must include:

- A 1-2 sentence summary of the method's purpose
- @param entries for all parameters, including what they are used for
- @return entry with a 1-2 sentence description of the return value
- @throws entries for any exceptions thrown.

The documentation for classes must includes:

- A 2-3 sentence description of the class' purpose
- @author the code's author
- @version a version for the class. Update on every commit to the repository.

2 Router Tool

The router tool must listen for incoming connections from clients and answer them in a multi-threaded architecture. That means the router can handle multiple client connections at once and respond to them appropriately.

The router takes the following parameters from the command line:

table Name/path of the file that contains the routing table to use for the router. The format for the routing table is shown below.

ip It must be an IPv4 address written in the typical four byte format (*e.g.* 10.10.10.10).

port The port to listen on. It must be an integer value [1025, 65535). (*e.g.* 5000, 6000).

2.1 Illegal parameters and addresses

If any of the parameters are missing or illegal, quit with an error message followed by a usage message. If the table contains an illegal or incorrectly formatted line, quit with an error message and a usage message.

2.2 Routing Table File Format

The routing table will be input to the program via a file name that is provided at the command line as a parameter. The format of the file shall be as follows:

SubnetNumber	SubnetMask	NextHop
...	...	...

Fields are delimited by spaces (there may be one or more spaces between fields). For example, the following is a potential routing table:

21.0.58.144	255.255.255.240	R1
21.0.58.0	255.255.255.0	R2
21.0.56.0	255.255.248.0	R3
21.0.48.0	255.255.240.0	R4
21.0.0.0	255.255.128.0	R5

Note that in the above table, the rows are ordered by subnet mask length. The input might be input in a non-sorted file. In such a case, you would need to first sort the table before you can use it. For instance, an equivalent input to the above table is:

21.0.48.0	255.255.240.0	R4
21.0.58.144	255.255.255.240	R1
21.0.0.0	255.255.128.0	R5
21.0.58.0	255.255.255.0	R2
21.0.56.0	255.255.248.0	R3

The tool must accept the table in any order and sort it appropriately if needed before use.

2.3 Router Algorithm

The router performs the following algorithm:

1. Open a TCP listening socket on the given IP address and port
2. Parse the routing table and stores the table's contents in a data structure in memory.
3. Listen for incoming network connections.
4. When a network connection arrives, spawn a thread to take care of the client connection.
5. Return to step 3 until the user presses 'q'.
6. When the user presses 'q', close the router and quit.

The spawned threads perform the following algorithm:

1. Receive an address from the client.
2. Look up the address in the routing table.
3. Find the most appropriate entry in the routing table. If multiple lines can match, choose the one with the longest subnet mask.
4. If a match is found, return the line's next hop address (*e.g.* R1).
5. If no match is found, return a message "NotFound"
6. If the address sent is not formatted correctly, send an error message. Example: Illegal address: 21.5.2
7. Return to step 1 until the client closes the connection
8. Close the TCP connection to the client.

The router thread and spawned threads output some log information about connections received and closed.

2.4 Router output formats

To make your programming go a bit faster, I provided String formats for all expected outputs in a class called `RouterConstants.java`. You are welcome to use them to make your outputs conform to the autograder's checks more easily.

2.5 Router Documentation (10 points)

Add Javadoc documentation to every method and class. The documentation for methods must include:

- A 1-2 sentence summary of the method's purpose

- @param entries for all parameters, including what they are used for
- @return entry with a 1-2 sentence description of the return value
- @throws entries for any exceptions thrown.

The documentation for classes must includes:

- A 2-3 sentence description of the class' purpose
- @author the code's author
- @version a version for the class. Update on every commit to the repository.

3 Test Cases (75 points)

There are several sets of test cases in the autograding.

3.1 Test Set 1: Small Table

In this set, the router is given the following routing table (short-router-table.txt):

```
21.0.58.144 255.255.255.240 R1
21.0.58.0 255.255.255.0 R2
21.0.56.0 255.255.248.0 R3
21.0.48.0 255.255.240.0 R4
21.0.0.0 255.255.128.0 R5
```

3.1.1 Router Test 1 (10 points)

The router's start up command is as follows:

```
java -jar Router-5784.jar -ip=127.0.0.1 -port=5050 -table=short-router-table.txt
```

After running the client commands below, the expected output for the router is:

```
Read 5 entries
Started listening on 127.0.0.1:5050
Press q to quit
Received connection from 127.0.0.1:34042
Closed connection from:  127.0.0.1:34042
Received connection from 127.0.0.1:34050
Closed connection from:  127.0.0.1:34050
Received connection from 127.0.0.1:34054
Closed connection from:  127.0.0.1:34054
Received connection from 127.0.0.1:34062
Received illegal address:  21.5.2
Closed connection from:  127.0.0.1:34062
```

The port numbers above will be different on your tests.

3.1.2 Client Tests (8 points)

The client will be tested against the router above. All tests must begin with `java -jar`. For brevity, that prefix is elided from all but the first test.

Test (1) `java -jar Client-5784.jar -dest=127.0.0.1 -port=5050 -addresses=21.0.58.150, 21.0.58.1,21.0.57.200,21.0.49.6,21.0.40.8,21.0.58.150,22.100.144.55`

Expected output:

```
Connected to router
21.0.58.150 -> R1
21.0.58.1 -> R2
21.0.57.200 -> R3
21.0.49.6 -> R4
21.0.40.8 -> R5
21.0.58.150 -> R1
22.100.144.55 not found
```

Test (2) `Client-5784.jar -dest=127.0.0.1 -port=5050 -addresses=21.126.128.5,85.249.95.171, 159.129.120.134,148.240.87.240,13.133.56.218`

Expected output:

```
Connected to router
21.126.128.5 -> R5
85.249.95.171 not found
159.129.120.134 not found
148.240.87.240 not found
13.133.56.218 not found
```

Test (3) `Client-5784.jar -dest=127.0.0.1 -port=5050 -addresses=205.181.189.90,176.29.92.149, 81.212.167.197,150.224.181.232,21.12.33.113,52.138.121.125,21.64.59.209,133.126.27.255, 56.67.13.81,21.154.101.150`

Expected output:

```
Connected to router
205.181.189.90 not found
176.29.92.149 not found
81.212.167.197 not found
150.224.181.232 not found
21.12.33.113 -> R5
52.138.121.125 not found
21.64.59.209 -> R5
133.126.27.255 not found
56.67.13.81 not found
21.154.101.150 not found
```

Test (4) `Client-5784.jar -dest=127.0.0.1 -port=5050 -addresses=21.100.25.12,21.5.2`

Expected output:

```
Connected to router
21.100.25.12 -> R5
21.5.2 -> Illegal address:  21.5.2
```

3.2 Test Set 2: Large Table

In this set, the router is given the following routing table (long-router-table.txt). The table has 254 rows. The first 5 and last 5 rows are shown here.

```
96.0.0.0 224.0.0.0 R1
128.0.0.0 224.0.0.0 R2
160.0.0.0 224.0.0.0 R3
192.0.0.0 224.0.0.0 R4
16.0.0.0 240.0.0.0 R11
...
133.227.198.128 255.255.255.192 R252
160.27.215.192 255.255.255.192 R250
162.253.20.128 255.255.255.192 R254
181.153.80.128 255.255.255.192 R244
189.8.41.0 255.255.255.192 R251
```

3.2.1 Router Test 2 (10 points)

The router's start up command is as follows:

```
java -jar Router-5784.jar -ip=127.0.0.1 -port=5051 -table=long-router-table.txt
```

After running the client commands below, the expected output for the router is:

```
Read 254 entries
Started listening on 127.0.0.1:5051
Press q to quit
Received connection from 127.0.0.1:45534
Closed connection from:  127.0.0.1:45534
Received connection from 127.0.0.1:45544
Closed connection from:  127.0.0.1:45544
Received connection from 127.0.0.1:45556
Closed connection from:  127.0.0.1:45556
Received connection from 127.0.0.1:45566
Received illegal address:  1.1.1
Closed connection from:  127.0.0.1:45566
Received connection from 127.0.0.1:45582
Closed connection from:  127.0.0.1:45582
```

3.2.2 Client Tests (10 points)

The client will be tested against the router above. All tests must begin with `java -jar`. For brevity, that prefix is elided from all but the first test.

Test (5) `java -jar Client-5784.jar -dest=127.0.0.1 -port=5051 -addresses=85.249.95.171, 159.129.120.134,2.50.25.12,148.240.87.240,13.133.56.218,74.186.187.75`

Expected output:

```
Connected to router
85.249.95.171 -> R45
159.129.120.134 -> R30
2.50.25.12 not found
148.240.87.240 -> R9
13.133.56.218 -> R17
74.186.187.75 -> R65
```

Test (6) `Client-5784.jar -dest=127.0.0.1 -port=5051 -addresses=2.27.193.114,6.8.97.45, 118.138.154.79`

Expected output:

```
Connected to router
2.27.193.114 not found
6.8.97.45 -> R24
118.138.154.79 -> R5
```

Test (7) `Client-5784.jar -dest=127.0.0.1 -port=5051 -addresses=59.79.130.93,121.227.252.126,2.12.22.22,197.168.47.14,197.168.47.13`

Expected output:

```
Connected to router
59.79.130.93 -> R15
121.227.252.126 -> R16
2.12.22.22 not found
197.168.47.14 -> R51
197.168.47.13 -> R51
```

Test (8) `Client-5784.jar -dest=127.0.0.1 -port=5051 -addresses=205.181.189.90,176.29.92.149,81.212.167.197,1.1.1,150.224.181.232,89.12.33.113,52.138.121.125,58.64.59.209,23.126.27.255,56.67.13.81,105.154.101.150`

Expected output:

```
Connected to router
205.181.189.90 -> R4
176.29.92.149 -> R10
81.212.167.197 -> R8
1.1.1 -> Illegal address:  1.1.1
150.224.181.232 -> R9
89.12.33.113 -> R22
52.138.121.125 -> R40
58.64.59.209 -> R52
23.126.27.255 -> R11
56.67.13.81 -> R76
105.154.101.150 -> R27
```

Test (9) `Client-5784.jar -dest=127.0.0.1 -port=5051 -addresses=68.189.142.23,190.179.212.203,17.70.51.141,9.255.108.148,189.160.222.218,196.81.140.48,1.210.190.23,218.18.55.31,48.199.220.76,220.27.167.223,110.80.10.153,90.98.72.196,43.96.28.233,195.213.11.44,67.57.128.177,99.207.130.137,37.6.69.115,8.168.165.181,159.136.68.67,47.176.108.153,77.92.255.57`

Expected output:

```
Connected to router
68.189.142.23 -> R23
190.179.212.203 -> R35
17.70.51.141 -> R11
9.255.108.148 -> R17
189.160.222.218 -> R37
196.81.140.48 -> R34
1.210.190.23 not found
218.18.55.31 -> R43
48.199.220.76 -> R6
```

```
220.27.167.223 -> R29
110.80.10.153 -> R1
90.98.72.196 -> R22
43.96.28.233 -> R14
195.213.11.44 -> R4
67.57.128.177 -> R23
99.207.130.137 -> R1
37.6.69.115 -> R32
8.168.165.181 -> R17
159.136.68.67 -> R30
47.176.108.153 -> R14
77.92.255.57 -> R19
```

3.3 Test Set 3: Large Table and Multi-threading

In this set, the router is given the same routing table as the previous set (long-router-table.txt). In this case, the router will be tested against a client that I wrote and provided for you in the starter code repository. The client is written in C and performs the following tasks:

1. Opens 3 sockets simultaneously to the router.
2. Sends an address to the router on socket 1. It waits for the router's response.
3. Sends an address to the router on socket 2. It waits for the router's response.
4. Sends an address to the router on socket 3. It waits for the router's response.
5. Repeats steps 2-4 until all addresses provided have been sent.
6. Closes all sockets and quits.

The purpose of the tests is to ensure that the router is multithreaded and can handle multiple conversations at once.

3.3.1 Router Test 3 (10 points)

The router's start up command is as follows:

```
java -jar Router-5784.jar -ip=127.0.0.1 -port=5052 -table=long-router-table.txt
```

After running the client commands below, the expected output for the router is:

```
Read 254 entries
Started listening on 127.0.0.1:5052
Press q to quit
Received connection from 127.0.0.1:50984
Received connection from 127.0.0.1:51000
Received connection from 127.0.0.1:51008
Received illegal address:  1.1.1
Closed connection from:  127.0.0.1:50984
Closed connection from:  127.0.0.1:51008
Closed connection from:  127.0.0.1:51000
```

The ports in your outputs will be different.

3.3.2 Client Tests (10 points)

The provided client will be tested against the router above. Note the use of the provided program (MultiClient) and that the program is not a Java program. The program receives 5 parameters - the router's IP address, the router's port, and three lists of addresses to sent (comma separated).

You **do not** need to modify the provided code at all. It should run as is.

Test (10) `./MultiClient 127.0.0.1 5052 85.249.95.171,159.129.120.134,6.9.22.9,22.23.3.2,1.1.1,148.240.87.240,13.133.56.218,74.186.187.75 85.249.95.171,159.129.120.134,148.240.87.240,2.33.56.218,74.186.187.75,21.5.10.6 59.79.130.93,1.23.23.0,121.227.252.126,197.168.47.13`

Expected output:

```
Socket 1 Response:  85.249.95.171 -> R45
Socket 2 Response:  85.249.95.171 -> R45
Socket 3 Response:  59.79.130.93 -> R15
Socket 1 Response:  159.129.120.134 -> R30
Socket 2 Response:  159.129.120.134 -> R30
Socket 3 Response:  1.23.23.0 -> NotFound
Socket 1 Response:  6.9.22.9 -> R24
Socket 2 Response:  148.240.87.240 -> R9
Socket 3 Response:  121.227.252.126 -> R16
Socket 1 Response:  22.23.3.2 -> R11
Socket 2 Response:  2.33.56.218 -> NotFound
Socket 3 Response:  197.168.47.13 -> R51
Socket 1 Response:  1.1.1 -> Illegal address:  1.1.1
Socket 2 Response:  74.186.187.75 -> R65
Socket 1 Response:  148.240.87.240 -> R9
Socket 2 Response:  21.5.10.6 -> R11
Socket 1 Response:  13.133.56.218 -> R17
Socket 1 Response:  74.186.187.75 -> R65
```

3.4 Test Set 4: Router Parameter Testing (11 points)

The following tests check the router's parameter parsing and error messages. They should all lead to the router not listening.

Test (11) Router Fail Missing Parameter 1

```
java -jar Router-5784.jar -port=5052 -table=long-router-table.txt
```

Expected output:

```
Missing parameter
Usage:  Router-5784 -table=t -ip=ip -port=p
```

Test (12) Router Fail Missing Parameter 2

```
java -jar Router-5784.jar -ip=127.0.0.1 -table=long-router-table.txt
```

Expected output:

```
Missing parameter
Usage:  Router-5784 -table=t -ip=ip -port=p
```

Test (13) Router Fail Missing Parameter 3

```
java -jar Router-5784.jar -ip=127.0.0.1 -port=5052
```

Expected output:

```
Missing parameter
Usage:  Router-5784 -table=t -ip=ip -port=p
```

Test (14) Router Fail Illegal Parameter 1

```
java -jar Router-5784.jar -ip=127.0.0.1 -port=abab -table=long-router-table.txt
```

Expected output:

```
Read 254 entries
Error parsing port:  For input string:  "abab"
Usage:  Router-5784 -table=t -ip=ip -port=p
```

Test (15) Router Fail Illegal Parameter 2

```
java -jar Router-5784.jar -ip=abcdef -port=5053 -table=long-router-table.txt
```

Expected output:

```
Read 254 entries
Error with IP address:  abcdef:  Name or service not known
Usage:  Router-5784 -table=t -ip=ip -port=p
```

The error message after the second colon is the one generated by Java's Unknown Address Exception and may be different on your computer.

Test (16) Router Fail Illegal Parameter 3

```
java -jar Router-5784.jar -ip=127.0.0.1 -port=5053 -table=nonexistentfile
```

Expected output:

```
Error reading table file:  nonexistentfile (No such file or directory)
Usage:  Router-5784 -table=t -ip=ip -port=p
```

3.5 Test Set 5: Client Parameter Testing (6 points)

The following tests check the client's parameter parsing and error messages. They should all lead to the client not connecting to the router.

Test (17) Client Fail Missing Parameter 1

```
java -jar Client-5784.jar -port=5051 -addresses=68.189.142.23,190.179.212.203
```

Expected output:

```
Error missing parameter
Usage:  Client-5784 -dest=ip -port=p -addresses=adds
```

Test (18) Client Fail Missing Parameter 2

```
Client-5784.jar -dest=127.0.0.1 -addresses=68.189.142.23,190.179.212.203
```

Expected output:

```
Error missing parameter
Usage:  Client-5784 -dest=ip -port=p -addresses=adds
```

Test (19) Client Fail Missing Parameter 3

```
Client-5784.jar -dest=127.0.0.1 -port=5051
```

Expected output:

```
Error missing parameter
Usage:  Client-5784 -dest=ip -port=p -addresses=adds
```

Test (20) Client Fail Illegal Parameter 1

```
Client-5784.jar -dest=abcdef -port=5051 -addresses=68.189.142.23
```

Expected output:

```
Error:  Illegal IP address:  abcdef:  Temporary failure in name resolution
Usage:  Client-5784 -dest=ip -port=p -addresses=adds
```

The error message after the second colon is the one generated by Java's Exception and may be different on your computer.

Test (21) Client Fail Illegal Parameter 2

```
Client-5784.jar -dest=127.0.0.1 -port=cats -addresses=68.189.142.23
```

Expected output:

```
Error:  Illegal port number:  For input string:  "cats"
Usage:  Client-5784 -dest=ip -port=p -addresses=adds
```

Test (22) Client Fail Illegal Parameter 3. Note that in this case, there is no router listening on port 6000 at 127.0.0.1.

```
Client-5784.jar -dest=127.0.0.1 -port=6000 -addresses=68.189.142.23
```

Expected output:

```
Error communicating:  Connection refused
```

4 Sample output files

I have provided a sample set of output files in the starter repository to help you compare your output.