Homework for Multithreading

Assignment:

Bryce Canyon Hiking Company is no longer satisfied with my implementation of the Socket Server that you connected your client to in the earlier homework. It wants you to develop your own server to provide the price quotes yourself.

In this homework, you are to build a socket server that will work with your original client application. It should be multi-threaded. That means, it should be able to process requests from multiple clients at any one time without having to queue them up and wait for others requests to process. Additionally, it should not crash when bad data is given to it, that means that you will need to check for:

- wrong number of arguments
- wrong type of arguments
- bad data in arguments

You will also need to provide specific feedback when something goes wrong (the text after the cost in the reply). Provide specific details about the errant input to help inform the user how to rectify the request. Don't just provide generic messages like "bad data". Keep in mind that the BhcUtils API already provides many (but not all) of the errors that you will report on, so you will just need to pass them in your server. Your job as a developer when building a server is to figure out how a user can break your software, and then defend against it.

Each student has been (or will be) assigned a port number for the server in a Canvas announcement. If you do not have a port assigned on the class computer at the time of this assignment, you can work on your local implementation until they are announced.

You should accept data in the form of:

```
hike_id:begin_month:begin_day:begin_year:duration:number_hikers (e.g: 0:8:1:2024:4:3)
```

The year input will have four digits, and all values are separated by colons (":"). You will also need to report errors for the duration if it's invalid for the given hike based on previous assignments. For this requirement assume the Hikes are associated with integer ids as provided by the enumeration in the API. Note that the ordinal() method of Java's Enum object will return a constant starting at 0 related to the specific Hike's position in the enum declaration.

Please note that the overall input to your program over the socket needs to be a String.

The returned result will be the cost followed by a colon (":"), followed by some text. If things go well, you'll give the cost and the text "Quoted Rate", if there is a problem, the cost must be set to -0.01 and the text will have an explanation of all the problems with the input. Again, note that the BhcUtils API already provides you with much of this capability. You should use the GUI from your earlier client homework to test your server. DO NOT INCLUDE A "GREETING" LINE in the returned data (like you have seen is some of the lecture examples), just accept the connection from the client, get the client input and return the answer or errors. That is the stated protocol and any changes to that will break the client-server relationship. I will be testing with a client built to that protocol an if your protocol is not exactly what I've specified (you put something else in, you aren't supplying the required response or meeting the defined protocol) then your server will not work with my client and that will result in penalties on your assignment.

As in the previous assignments, you will need to use the provided jar file (API) to assist you in your project. You can download the package with the jar file and a compressed folder with the html JavaDocs (API documentation) for the classes in the jar here and the API documentation is provided in the download link above, or you can view it on the class site here. You can also view the original description of the api from the week 5 assignment here.

As suggested, you should use your earlier client to test your server, though you could also use telnet as demonstrated in our lecture material. However, I will be using my own client to test your app, so DO NOT include client code with your homework submission.

Test Cases

Here is a minimal set of test cases that you must handle correctly in your code. I will leave it to you to determine whether the result should be valid or invalid (some should be obvious) based on the API. Depending upon your choice of input I will also be testing other forms of errant input as well, like missing fields or bad data (characters for numbers, etc). Note that your server should return error messages as expected based on the table below.

Hike	Duration	Start Of Tour	Number of Hikers	Expected Result
Navajo Loop	4 days	July 1, 2025	2	320.0
Navajo Loop	9 days	July 1, 2025	1	9 is not a valid duration for selected hike The ending date was not defined
Navajo Loop	4 days	January 1, 2025	2	The starting date is out of season The ending date is out of season
Navajo Loop	4 days	July 1, 2023	2	Begin date for hike has already past
Navajo Loop	4 days	Last Valid Date of Season	1	The ending date is out of season
Navajo Loop	4 days	July 1, 2100	1	Year falls outside valid range
Navajo Loop	4 days	July 55, 2025	1	Date and month combination is not valid
Navajo Loop	4 days	July 1, 2025	-3	Number of hikers [-3] cannot be less than 1
Navajo Loop	4 days	July 1, 2025	500	Number of hikers [500] is over the limit of #
The following test cases will be dependent upon UI limitations				
Bad text in some or all fields (aka, alpha chars instead of numbers)				Error message provided by you
Data missing in fields				Error message provided by you

Submission:

1. Get your project working on your local machine
2. Use the previous instructions here, to create an executable jar file to upload to the class server.
3. Upload the jar file to the class server. Place it in your home directory (not in your public_html folder).
4. You can run your app by typing java -jar <jar name> at the command line. To kill the server, just type Ctrl-C.
5. When you are content with your submission, run it by typing the following at the command prompt:

   ```
   nohup java -jar <jar name> &
   ```

   If you make a mistake, you will need to get he process id of the app you started. Type:

   ```
   ps -ef | grep <your last name>
   ```

   To find your processes and get the process id from the output. Then run:

   ```
   kill -9 <process id>
   ```

   to kill it
6. Zip your source code and upload it to Canvas using the naming convention from the class syllabus. In the comments section make sure you put **the port number you used so I can test the code.**

Grading:

I will use my own test client to test your server. Your server **MUST** be up and running on the class server for me to test against. **DO NOT** submit your test client.

Use the following breakdown as a GUIDELINE ONLY for general weighting of different parts of the assignment. Specific grading may not completely fall into the breakdown below:

1. Reasonable code design/Coding Guidelines/General Grading Guidelines 25%
2. Error Messages (Complete and Accurate) from the server 25%
3. Application runs and passes all test cases laid out above 30%
4. Multi-threaded implementation 15%
5. Port number included in homework submission 5%