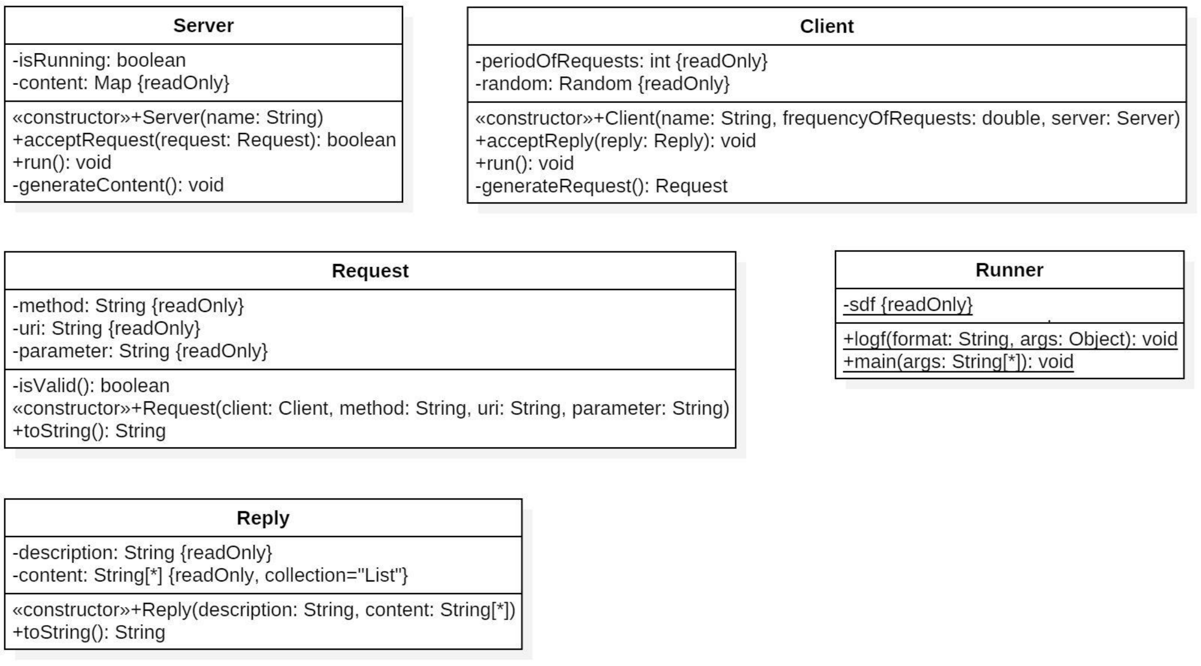
CENG 443 – Object-Oriented Programming Languages and Systems

Spring 2017 - Homework 2

Client – Server Simulation

Selim Temizer

Feedback : Between May 22nd and May 26th, 2017
Due date : May 31st, 2017 (Submission through COW by 23:55)



Server
-isRunning: boolean
-content: Map {readOnly}
«constructor»+Server(name: String)
+acceptRequest(request: Request): boolean
+run(): void
-generateContent(): void
Client
-periodOfRequests: int {readOnly}
-random: Random {readOnly}
«constructor»+Client(name: String, frequencyOfRequests: double, server: Server)
+acceptReply(reply: Reply): void
+run(): void
-generateRequest(): Request
Request
-method: String {readOnly}
-uri: String {readOnly}
-parameter: String {readOnly}
-isValid(): boolean
«constructor»+Request(client: Client, method: String, uri: String, parameter: String)
+toString(): String
Runner
-sdf {readOnly}
+logf(format: String, args: Object): void
+main(args: String[*]): void
Reply
-description: String {readOnly}
-content: String[*] {readOnly, collection="List"}
«constructor»+Reply(description: String, content: String[*])
+toString(): String

In this homework, we will build a simplified simulation of the request and reply (response) messages being exchanged between a server and a few clients. Stub code is provided in the homework bundle for all the required classes. You will need to fill in ONLY the parts marked with the word "***TODO***". Do not modify any other parts, and do not add any other classes. Output from sample runs are also provided. Try to match the output format as closely as possible.

The classes that we will need are briefly as follows:

- ***Server***: There will be a single instance of this Thread subclass, and it will represent a server getting requests from clients, and answering them properly. Within the server code, there is a timer that is supposed to shut down the server after a predefined time period.
- ***Client***: There will be multiple instances of this Thread subclass at run time. This class will represent a client which first sends a GET request to fetch the *Index* of the contents present at the server, and then randomly generates GET or POST requests at predefined intervals until the server shuts down.

- ***Request***: Can be a GET or POST request with other additional fields. The clients will be passing request instances to the server thread.
- ***Reply***: Represents a response message originating at the server and being passed from the server to the client (which generated the request leading to this response message).
- ***Runner***: This is the runnable class that initializes and starts the simulation. It also contains an optional static utility function, that is similar to the ***printf*** function, in order to print timestamped text on the screen (you may choose not to use this function, and implement your own timestamped printing routines).

When the timer fires, the server should not accept any other requests from the clients (and therefore, the clients will figure out that the server is about to shut down). Then, all threads will stop what they are doing, print brief reports, and stop gracefully. Note that each line of the reports should be printed by a separate printing statement, and no output should be garbled (i.e., threads will also need to synchronize their printing).

When implementing the ***Client*** and the ***Server***, you should only use ***primitive threading constructs*** (do not use high level concurrency utilities). Also, you may need to access some fields and/or methods from the ***Request*** and ***Reply*** classes, but the developer of those classes chose to make most of the members private, and did not provide getters/setters. Therefore, you should use ***reflection*** in order to access the required members.

What to submit? (Use ***only ASCII characters*** when naming all of your files and folders)

1. Any possible documentation that you may have (in a directory named "***Docs***").
2. Your source code (all in a single directory named "***Source***"). Do NOT submit binary code or IDE created project files. Just submit your ".java" files (all 5 files). We should be able to compile and run your code simply by typing the following:

```
C:\...\Source>  javac  *.java
C:\...\Source>  java  Runner  <parameters>
```

Zip the 2 items above together, give the name <ID>_<FullName>.zip to your zip file (tar also works, but I prefer Windows zip format if possible), and submit it through COW. For example:

e1234567_SelimTemizer.zip

You are welcome to consider implementing some visual extensions and/or creative additions to the baseline work described above. There will be bonuses awarded for all types of extra effort. If your bonus work requires significant modification to the stub code, please discuss your ideas with the instructor before implementing them. Late submissions will NOT be accepted, therefore, try to have at least a working baseline system submitted on COW by the deadline. Good luck.

IMPORTANT: Late submissions (even for 1 minute) will not be accepted!

We will only grade submissions on COW, and system closes automatically at due time!