Distributed Systems Project 1
22/April/2020

Prelude

- To better present the assignment, we will imagine a person who is the project owner of this assignment.
- The word “clients” is reserved to refer to the DictionaryClient software.

Problem Context:

The project owner wants to create a dictionary application, and he has the following specifications:

1. The application needs to be platform independent.
2. The app can do the basic add, remove, and query operations of a dictionary.

Some non-functional requirements would include

1. Good failure model
    a. Failure of clients does not affect each other.
    b. Client is resilient to server failure.
2. High transparency.
    a. Reliable connection.
    b. Minimal setup
3. Intuitive interface.
4. Fast response from server.

The project has the following constraints:

1. Limited amount of network and computing power.
2. Must be developed using Java.
3. Must use multi threading and socket programming.

System Design

Class diagrams and sequence diagram can be found at the end of the document.

Design of the protocol

To handle different types of requests, I created class objects for requests, responses, database access. Each of these objects contains a status field of some enumerable values. Instead of using exceptions to capture errors, the status field is used to provide more descriptive responses.

Design the server

The server uses thread-per-connection architecture, therefore, each connection with the client is abstracted as a single object called “ClientConnection”.

The service provided by the server is abstracted in the “Service” interface. In the future, different services could be provided to clients of different permission levels.

Design the client

The client also abstracts a connection as a single class called “ClientConnection”.

For the GUI, the client controls the display that holds the responsibility of displaying relevant data for clients. The grab data from the GUI, the client is a listener of the GUI class. The GUI is based on the JSwing library and designed using IntelliJ GUI designer.

Critical Analysis

Failure Models

TCP is used to handle send and receive omission failures.
The client requests a new socket when the existing socket timeout or the host becomes unreachable. This is good because when clients fail, the server can simply close the connection and wait for the client to reconnect. The use of timeout solves the crash and fail-stop failures. [1]

Artitecture:

A dynamic model of thread per connection is used. A timeout is used to free sockets from inactive clients.

Compared to other dynamic and static models, thread per connection has its pros and cons. This is a better design option than thread per request because client's requests are chronologically in nature. The benefits of a dynamic model is that it only uses the number of threads needed. Static models such as thread pools continue to use a significant amount of sockets even on standby. However, when a lot of connections get closed and opened, the creation of new sockets would be expensive, whereas static models avoid this problem by not closing and reusing sockets.

A better model would be a hybrid model that is dynamic on small loads but static on heavy loads. This would be suited for real use cases when dictionary services experience a spike during exam seasons. [2]

Data verification

All data verification is done on the server end, which is more secure than client end verification. Server end verification includes checking for null and empty requests, which isn't computationally heavy. Non-words aren't picked up by the servers, and it will happily accept "4EVER" as a real word. I regard this as an advantage of the system, because the client has the freedom to define new worlds, and to add non-English words.

The drawbacks to the lack of data verification is the possibility of typos. To combat this problem, the history of edits is temporarily saved, and the client can see a list of all words in the dictionary.

Database:

The dictionary is a java map object of words to meanings. This has the limitation of not allowing fuzzy search, nor search by definitions.

The use of java serialized objects to store the dictionary forces the server to speak to java programs, which was a problem if I were to automate debugging using netcat. The serialized object needs to be overwritten to disk instead of appending which becomes inefficient for large dictionaries. Because the whole dictionary needs to be loaded into memory, it becomes infeasible to create large dictionaries.

The use of “Data” objects to store data and the use of adapter patterns for “Connector” classes would make it easy to use a different data type. In fact, CSVConnector was initially used but lacked the ability to properly escape commas, and was left unused.

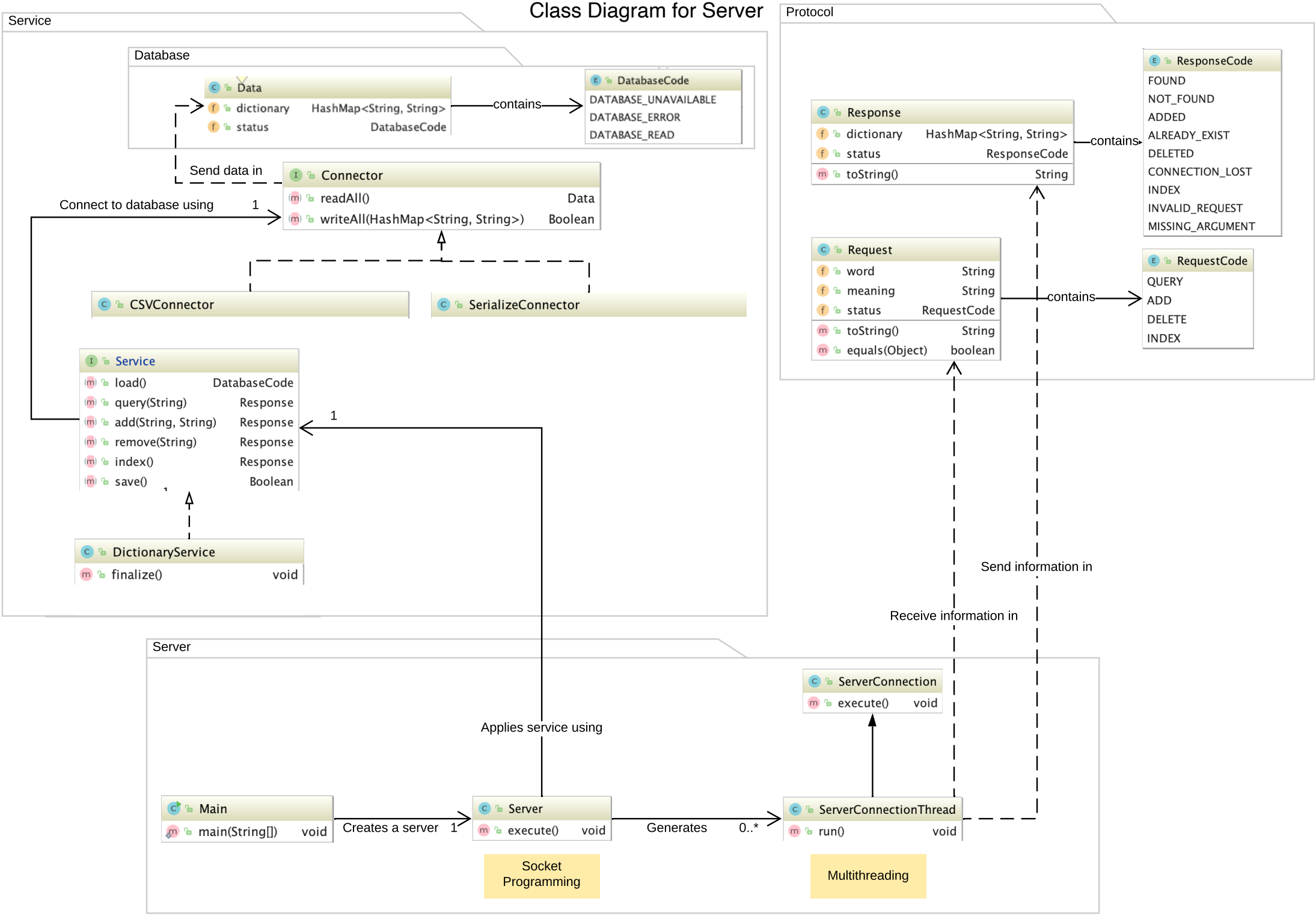
Conclusion. It would be best if the database could be connected to SQL or NOSQL server to allow faster save and search compared to native serialized objects in memory. N-grams and index on meaning would make fuzzy search possible.

References

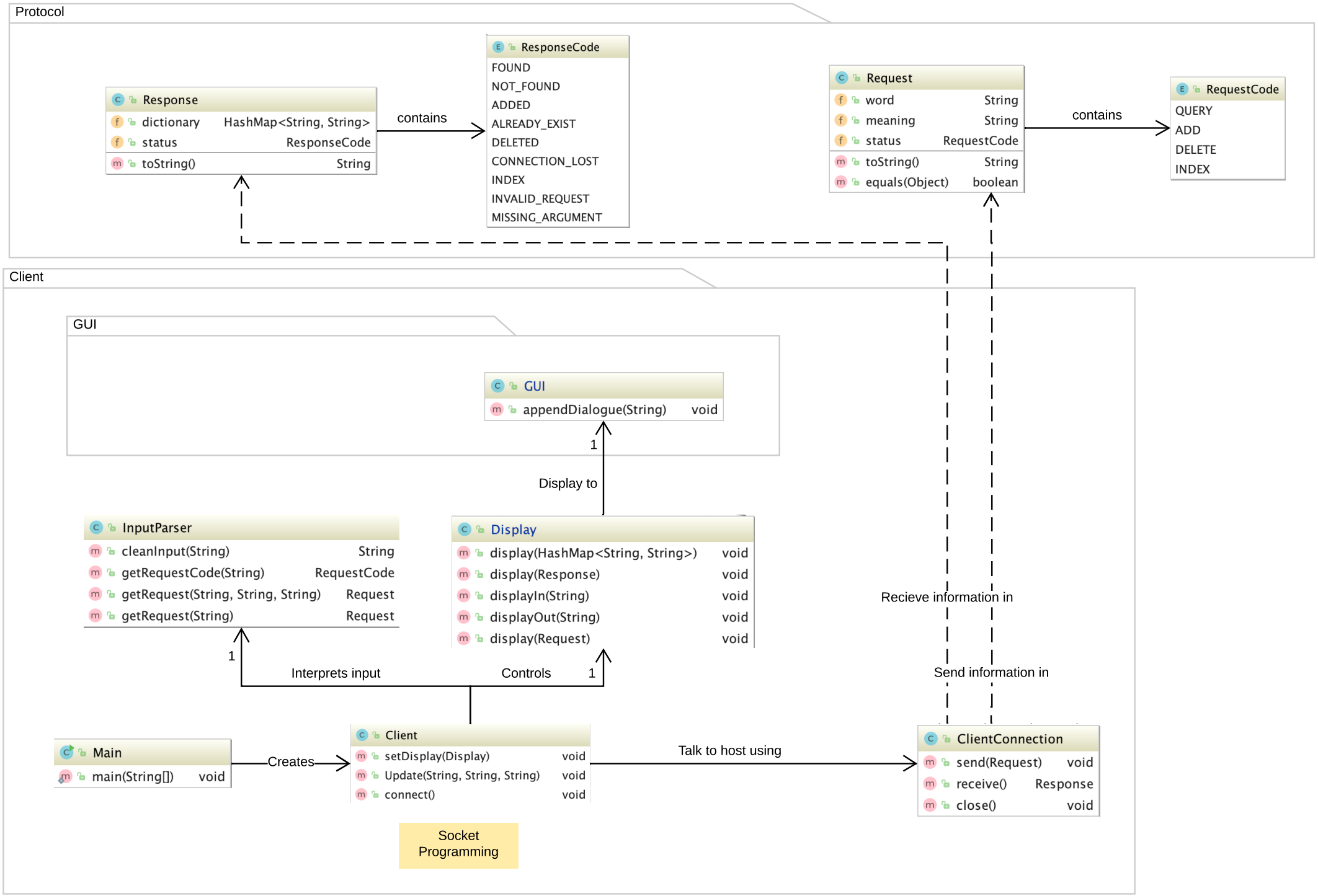
[1] G.Colulouris, J. Dollimore, T. Kindberg, G.Blair, Distributed Systems Concepts and Design, 5th Ed.

[2] Google tread on the word dictionary. Accessible: https://trends.google.com/trends/explore?geo=AU&q=dictionary

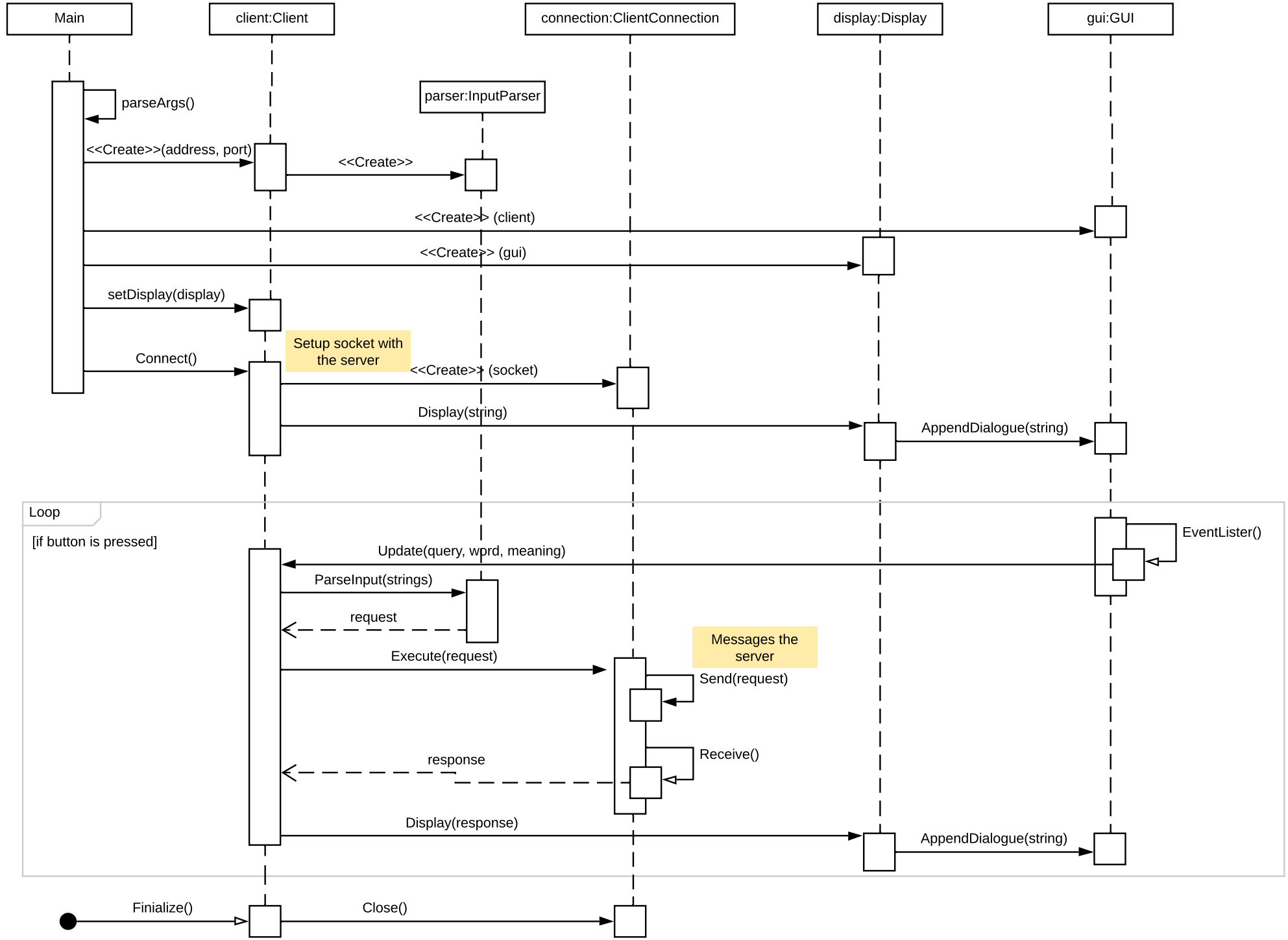
Class Diagram for Server

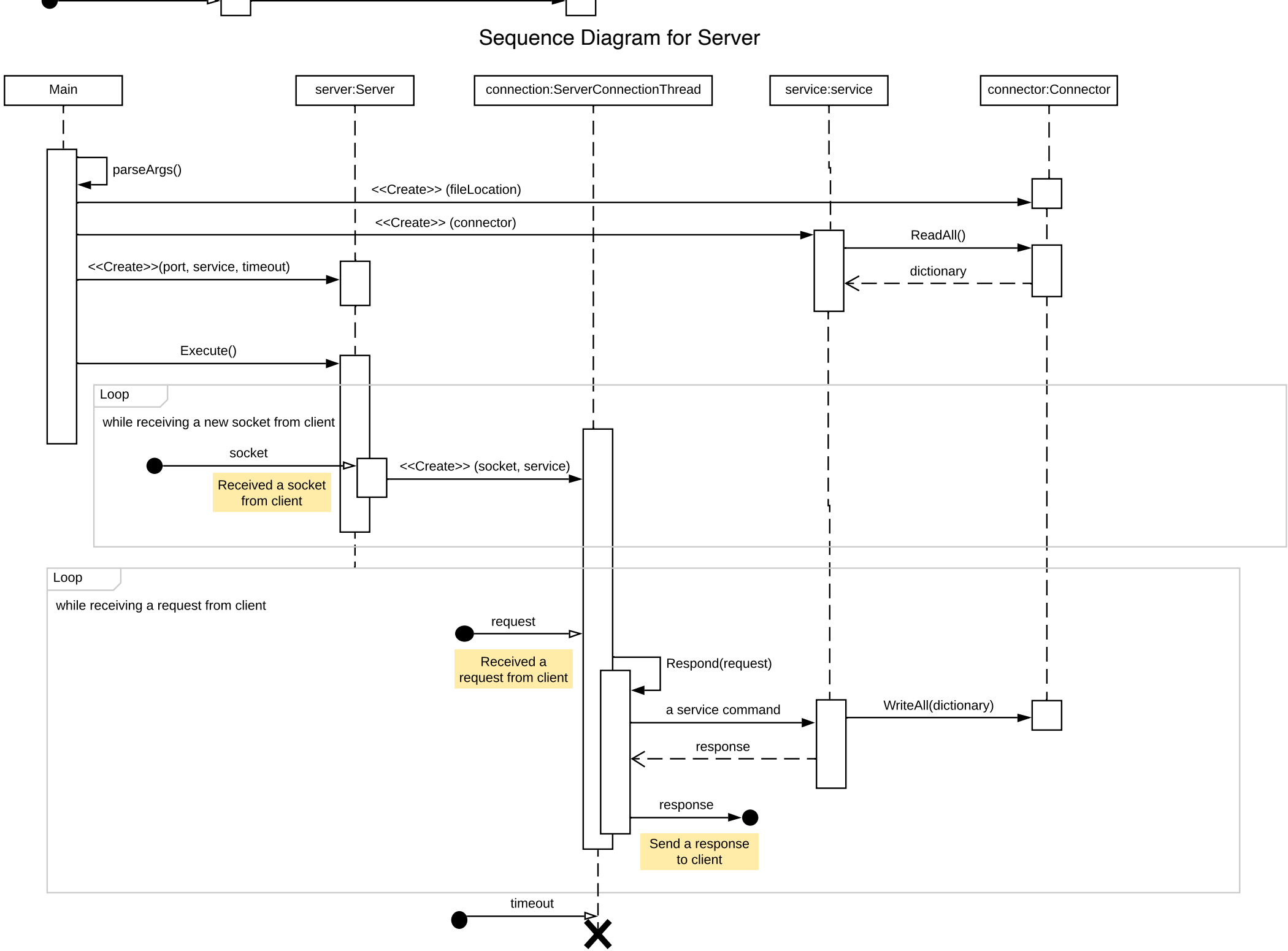


Class Diagram for Client



Sequence Diagram for Client




Sequence Diagram for Server
Main
server:Server
connection:ServerConnectionThread
service:service
connector:Connector
parseArgs()
<<Create>> (fileLocation)
<<Create>> (connector)
ReadAll()
dictionary
<<Create>>(port, service, timeout)
Execute()
Loop
while receiving a new socket from client
socket
Received a socket from client
<<Create>> (socket, service)
Loop
while receiving a request from client
request
Received a request from client
Respond(request)
a service command
WriteAll(dictionary)
response
response
Send a response to client
timeout