Name: Zixin Ye

Student ID 1101188

The main goal of our project is to build a digital dictionary which uses a client-server architecture and implements a multi-threaded server that allows concurrent clients to search the meaning(s) of a word, add a new word, and remove an existing word. We implement a thread-pooled server to allow concurrent clients.

We decide to implement a simple English dictionary, which means we only allow some simple English words to be added in our database. The maximum length of word we allow is fifteen and the maximum length of meaning we allow is eighty. No one can give a string longer than these because we simply set a restriction to the client GUI.

We also do not allow any other characters except English characters and whitespace. You cannot type anything other than English characters and whitespaces in our text field and text area. This is also a restriction we set to the client GUI.

The database we use is a MySQL database and we use Java JDBC connector to connect my server to a database. Therefore, there are some situations when a failure arises. First, if the JDBC connector has not been set in the Java Build Path, then our server would fail to connect the database. Second, if the database has not been set up, our server would fail. The database we need in the environment is:

1. name as dictionary,
2. allow anonymous user at localhost to select, insert and delete,
3. has a table called mainDictionary,
4. its mainDictionary table has two attributes word and meaning, and word must be distinct,
5. both two columns must be not null,
6. the maximum length of string in word attributes should be at least fifteen and the maximum length of string in meaning attributes should be at least eighty.

If one of the conditions is not met, then our server will fail. We will provide a MySQL script to create the database and a sql file, which we use for testing, to import at the submission.

Our system has two main parts, client and server. The client side has five java classes. The Client.java is the main class and it will build an OperationFrame, myGUI. OperationFrame will create the GUI and add all restrictions we mention above to the user interface. It will create a Jbutton, submit and our client will connect to the server each time when we click the submit button. The Asker class is used to build the connection to the server. It will tell the server what we want to do and receive the result from server. The InputRestriction class is what we use to prevent user entering invalid input. The CustomOutputStream class is what we use to redirect the System.out printStream. It will let the result print in the output text area instead of the command window.

The server side has six classes. The ThreadPooledServer is the main class and it will build a thread pool. The ThreadPooled class is the ThreadPool we build. We use LinkBlockingDeque data structure to implement a queue. The Worker class extends Thread. We set three Workers in a thread pool to handle the inputs, which means we will have three threads to handle the inputs. The Receiver class is used to receive the connections from clients. It will add the request to the queue. The DictOperation class is to send the required operation to the MySQLAccess class. The DictOperation class will analyse the result from MySQLAccess and catch any exceptions. The MySQLAccess is the class which do the operations directly to the database. It will throw the exceptions to the DictOperation and let it handle the exceptions.

The logic of our system is simple. We have a client side and a server side. The user interface in the client is very useful. It has a JComboBox to let us select what to do. The information we send to the server will be based on the operation we select on the JComboBox. If we do not select anything, the connection will not establish. Second, the text area under the JCombox is the output screen. We do not allow any editing on the text area, but you can click the clear button to clean the screen. The buttons below the output screen are submit button and clear button. As we mention before, the clear button is simply to clean the output screen. For the submit button, it will collect the information at the word text field and at the meaning text area. It will only collect the information in the meaning text area when you select “add” at the combo box. Otherwise, it will simply ignore the information there. If all your inputs do not violate the restrictions at the GUI, the client program will create a socket containing a “packed” string. The string will follow the format “operation:word:meaning” if the operation is “add”. Otherwise, the format should be “operation:meaning”. “Operation” at the first character is one of “+”, “-” and “=”. They represent “add a word”, “delete a word” and “show the meaning of a word” respectively. The parseExecution(String line) function in the server side will understand the format.

As for the server side of our system, it is also useful. The ThreadPooledServer class will initialize a ThreadPool object. The ThreadPool class will create three workers, which means three threads, and assign a task queue to each worker. Whenever a worker is available and the queue is not empty, workers will pop out a task fom the queue and use parseExecution() function to execute the input. The parseExecution() function will create a DictOperation object and send the message to the DictOperation object. The DictOperation class will then send the messge to MySQLAccess class, which can communicate with the database directly and throw any exceptions to the DictOperation class. The DictOperation class should be able to handle any exceptions from the database, which means it will always give a result to both the client side and the server side even if the result is an error message. However, we are afraid that there might be some unexpected errors so we use a catch(Exception e) clause to catch all unexpected errors and print the error message to both the clients and the server.

We also include the class diagrams and a sequence diagram for operations to illustrate our design at the attachment.

We think that our system works fine because:

1. We cannot detect any errors when we are testing, which means we at least finish the core of the project.
2. The GUI looks good and it is easy to use with labels at each component.

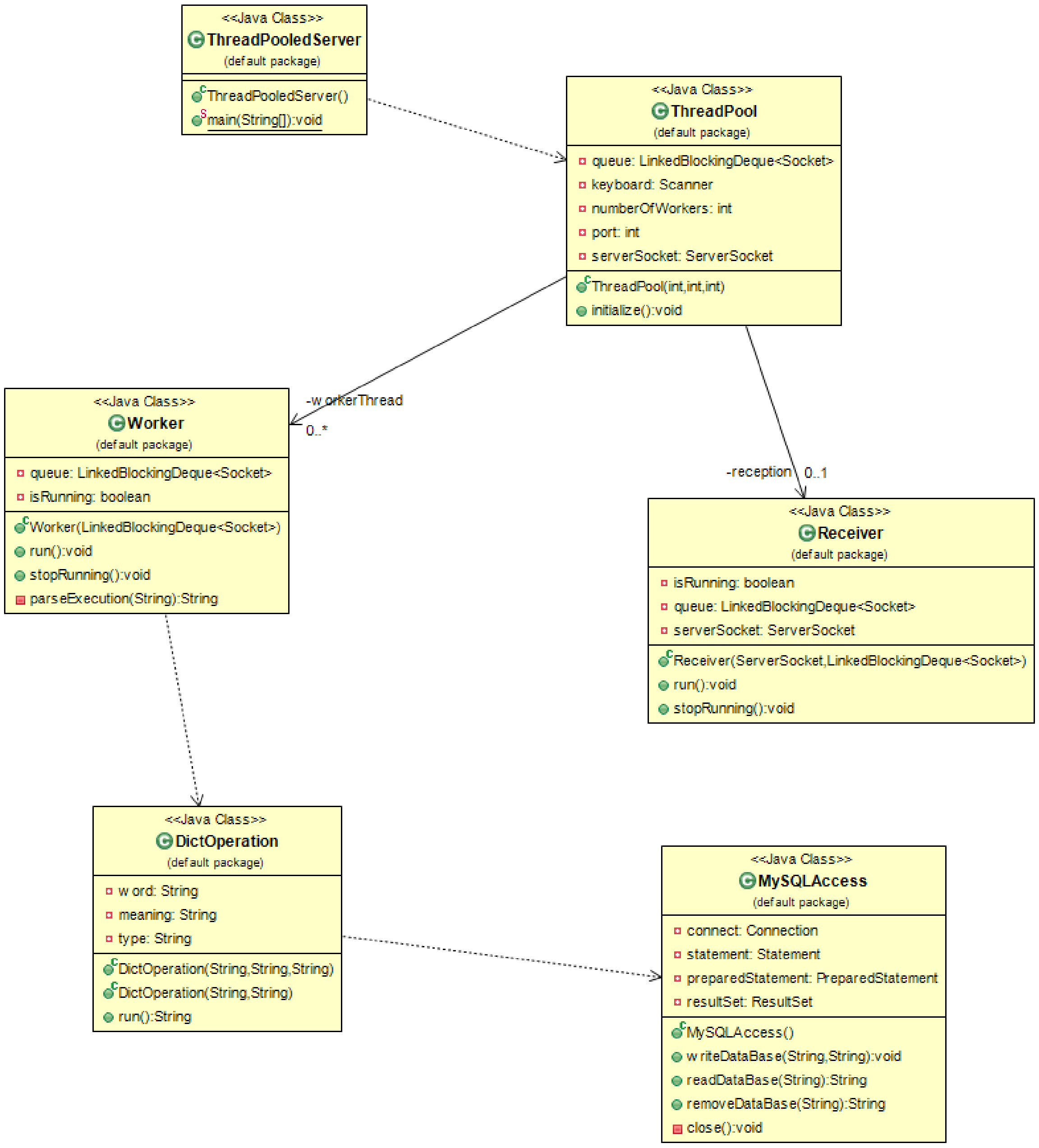
3. We also try our best to handle the errors such as preventing users from entering invalid input, which can also reduce the workload at server, and giving users proper error messages.
4. We also discuss that a thread pool is the most proper design in our circumstance. The reason is that we do not need to any long run operations, but we might need to handle a large number of short run operations in the server.

However, we argue that there are some further enhancements we can apply:

1. We probably need to add more functionalities to the server. The server only supports plain English words and the maximum length of words and meanings is too small.
2. Additionally, the server does not support multiple meanings for a same word is inconvenient. Therefore, some improvements can be applied to our server.
3. We have difficulty in deciding the size of queue in the thread pool. We think it is hard to find the ideal size of queues and the ideal number of threads. We are afraid that if we set the length of queue too long, then the waiting time for clients might be too long. However, considering that our operations are not time consuming, a long queue might not be a bad choice. Therefore, we think we might require more tests to estimate the expected waiting time for each user in different circumstances.
4. We require some configurations to the running system. As we mention at the first part of our report, we need to set up a specific database before we use the server. This will lose the flexibility and scalability of a distributed system should have. We might need to design a more flexible database connection.

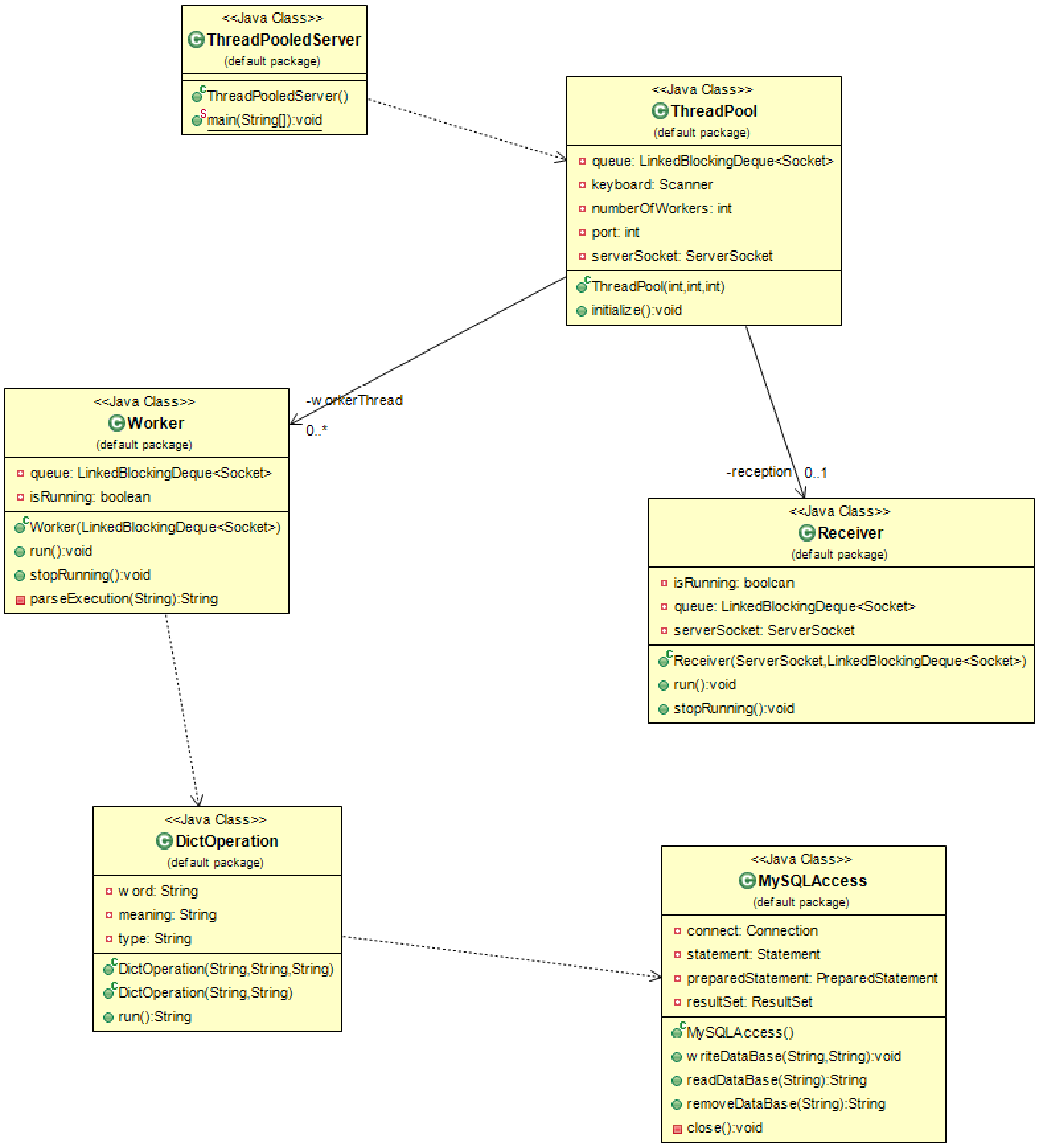
Creativity part:

1. We implement a customized thread pool by LinkedBlockingDeque.
2. Our user interface is easy to use and label each component.

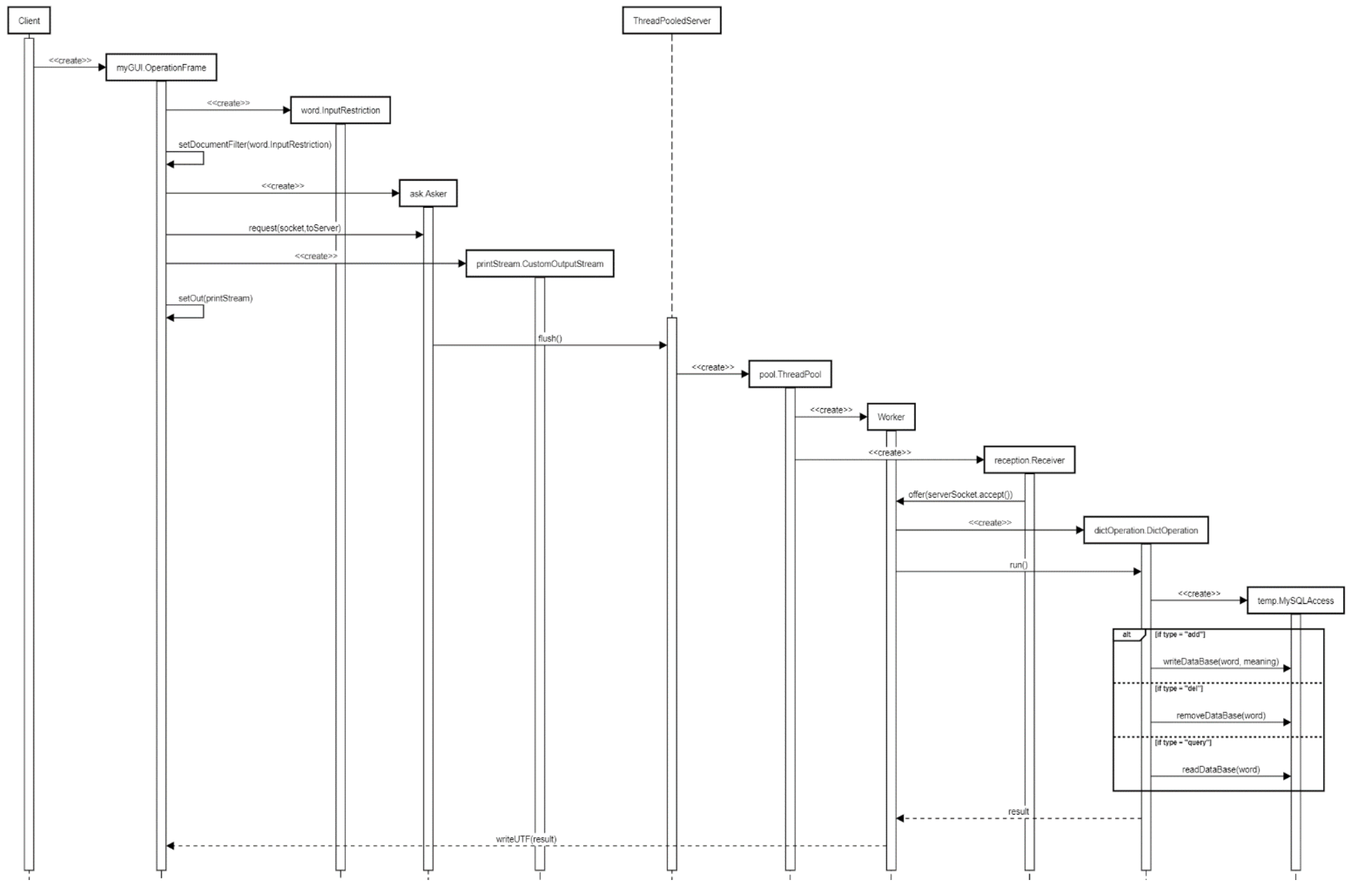


Graph 1: Class diagram of Server

Graph 2: Class Diagram of Client



Graph 3: Sequence Diagram of Operations


Client
ThreadPooledServer
<<create>>
myGUI.OperationFrame
<<create>>
word.InputRestriction
setDocumentFilter(word.InputRestriction)
<<create>>
ask.Asker
request(socket,toServer)
<<create>>
printStream.CustomOutputStream
setOut(printStream)
flush()
<<create>>
pool.ThreadPool
<<create>>
Worker
<<create>>
reception.Receiver
offer(serverSocket.accept())
<<create>>
dictOperation.DictOperation
run()
<<create>>
temp.MySQLAccess
alt
[if type = "add"]
writeDataBase(word, meaning)
[if type = "del"]
removeDataBase(word)
[if type = "query"]
readDataBase(word)
result
writeUTF(result)