COMP90015 Assignment 1 Report

Name: Duc Trung Nguyen

Username: ductrungn

Student Id: 1069760

1 Problem Context

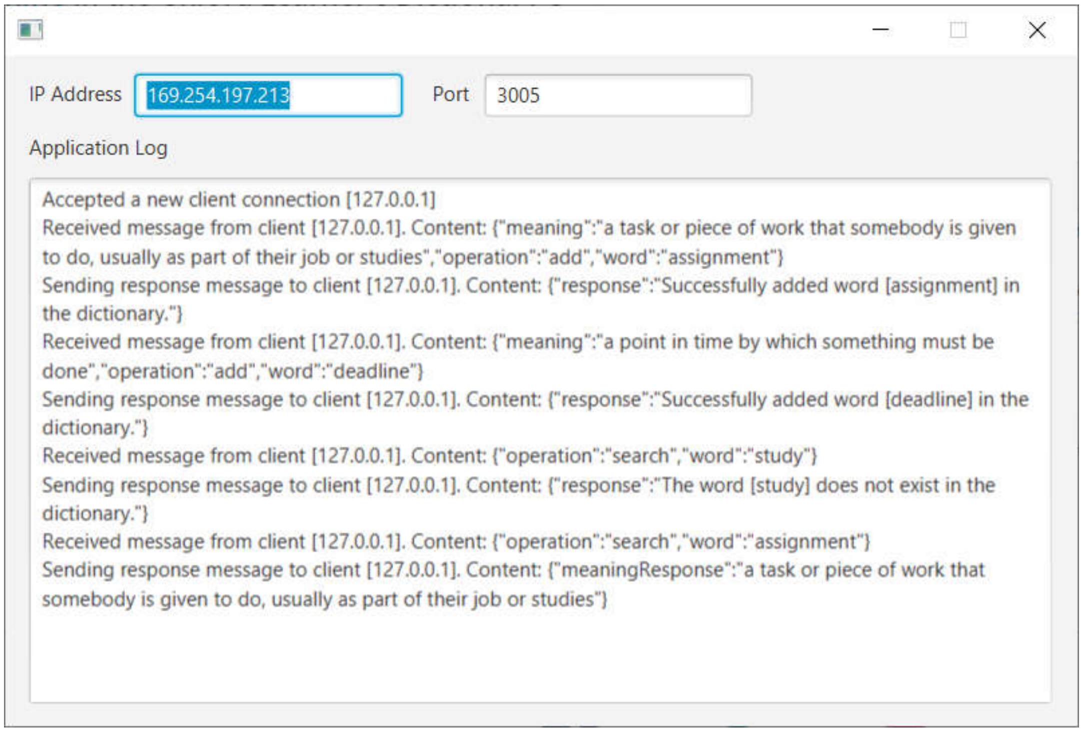
The objective of this assignment is to implement a multi-threaded dictionary server to serve concurrent requests from multiple clients at a time. The server supports the following services:

- Give the meanings of a word.
- Add a new word into the dictionary.
- Delete a given word.

The socket communication, threading, and synchronization technologies must be used to design server and client applications.

2 User Interface

2.1 Dictionary Server

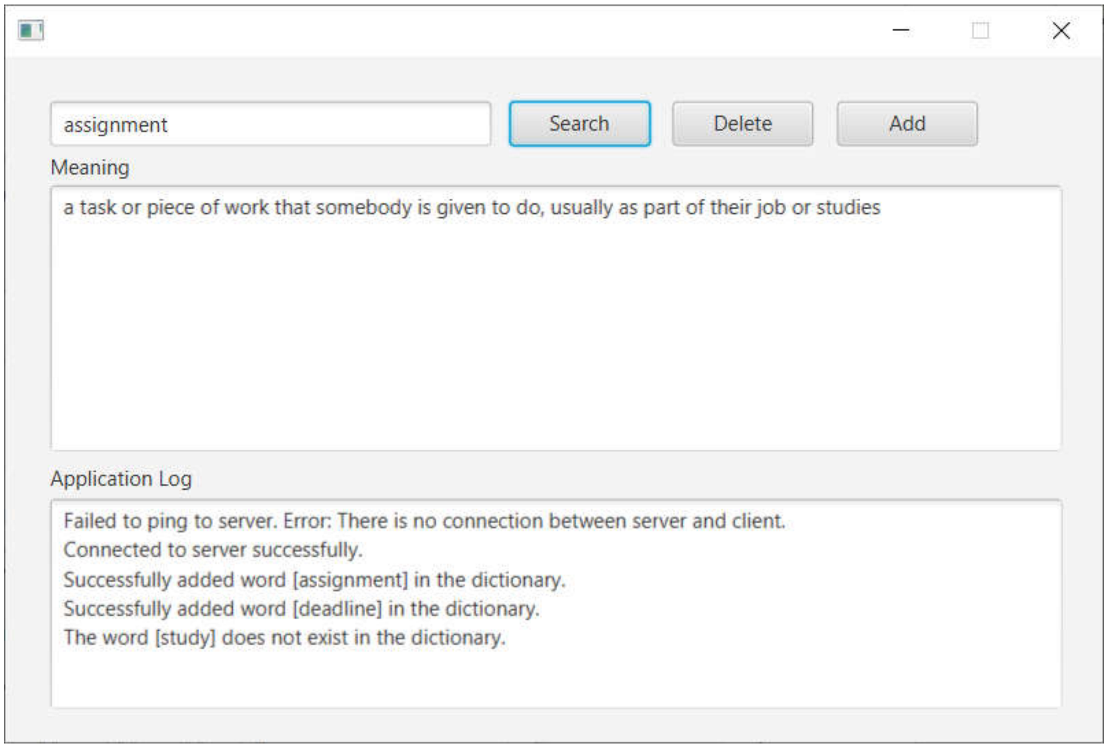


```
Accepted a new client connection [127.0.0.1]
Received message from client [127.0.0.1]. Content: {"meaning":"a task or piece of work that somebody is given to do, usually as part of their job or studies","operation":"add","word":"assignment"}
Sending response message to client [127.0.0.1]. Content: {"response":"Successfully added word [assignment] in the dictionary."}
Received message from client [127.0.0.1]. Content: {"meaning":"a point in time by which something must be done","operation":"add","word":"deadline"}
Sending response message to client [127.0.0.1]. Content: {"response":"Successfully added word [deadline] in the dictionary."}
Received message from client [127.0.0.1]. Content: {"operation":"search","word":"study"}
Sending response message to client [127.0.0.1]. Content: {"response":"The word [study] does not exist in the dictionary."}
Received message from client [127.0.0.1]. Content: {"operation":"search","word":"assignment"}
Sending response message to client [127.0.0.1]. Content: {"meaningResponse":"a task or piece of work that somebody is given to do, usually as part of their job or studies"}
```

Figure 1 The Severe Dictionary application

- IP Address text filed: show sever computer's IP address.
- Port text field: show port using to make socket communication between server and clients.
- Application Log text area: show all processing logs between server and clients.

2.2 Dictionary Client



a task or piece of work that somebody is given to do, usually as part of their job or studies

```
Failed to ping to server. Error: There is no connection between server and client.
Connected to server successfully.
Successfully added word [assignment] in the dictionary.
Successfully added word [deadline] in the dictionary.
The word [study] does not exist in the dictionary.
```

Figure 2 The Client Dictionary application

- Word text field: user must type a word before performing search, delete, or add operation.
- Meaning text area: show word meanings retrieved from the dictionary serer.
- Application Log text area: show all processing logs between server and clients.

3 Architecture

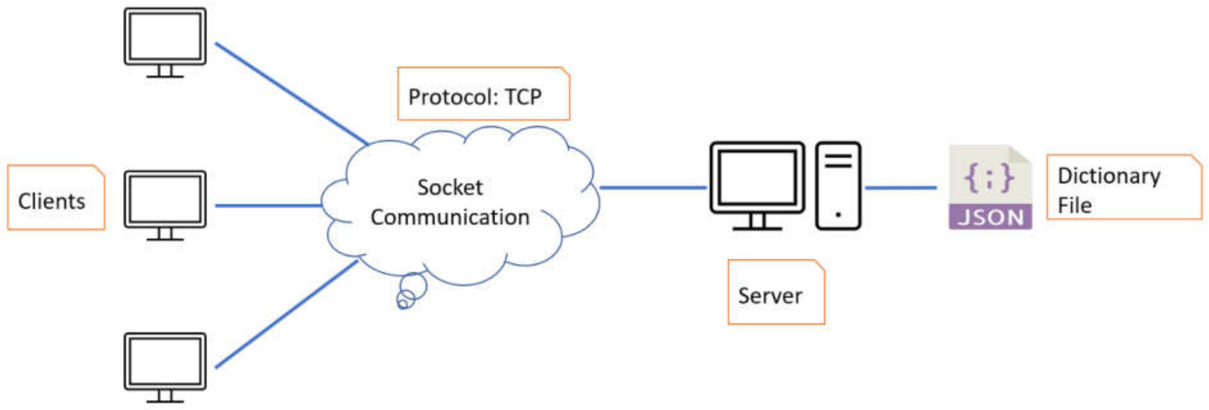


Figure 3 The system architecture

- Server load the dictionary configuration file in JSON format and keep it into memory during application lifecycle.
- Server only persist in-memory dictionary to the JSON file when it is shutdowned.
- Server accept and manage all client socket connections.

- Client connects to Server to perform search, add, and delete operations.
- The socket communication takes place between Server and Client through TCP protocol.

3.1 Dictionary Server

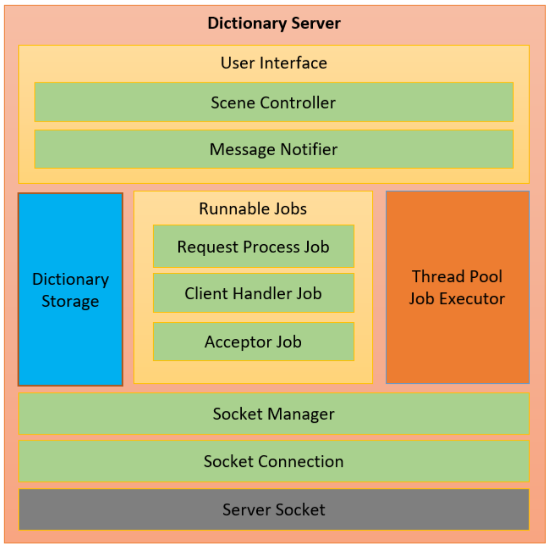


Figure 4 The Server Dictionary architecture

- **Scene Controller**: is responsible for manage user interface and handle user actions.
- **Message Notifier**: is responsible for append processing logs to Application Log text area.
- **Dictionary Storage**: is responsible for loading and persisting the dictionary file. It also supports add, delete, and search operations.
- **Thread Pool Job Executor**: is responsible for managing allocated threads and executing user defined jobs.
- **Request Process Job**: is responsible for processing all client requests.
- **Client Handler Job**: is responsible for handling the client socket communication.
- **Acceptor Job**: is responsible for accepting the client socket connections.
- **Socket Manager**: is responsible for managing all client socket connections.
- **Socket Connection**: is responsible for establishing the socket connection, sending and receiving messages through the socket input and output streams.

3.2 Dictionary Client

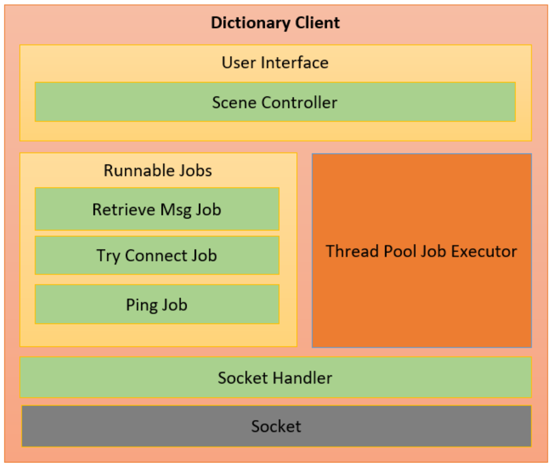


Figure 5 The Dictionary Client architecture

- **Scene Controller**: is responsible for manage user interface and handle user actions.
- **Thread Pool Job Executor**: is responsible for managing allocated threads and executing user defined jobs.
- **Retrieve Msg Job**: is responsible for retrieving the response message from Server.
- **Try Connect Job**: is responsible for re-establishing the socket connection to Server.
- **Ping Job**: is responsible for detecting the socket connection alive status.
- **Socket Handler**: is responsible for establishing the socket connection, sending and receiving messages through the socket input and output streams.

4 Class Diagram

4.1 Dictionary Server

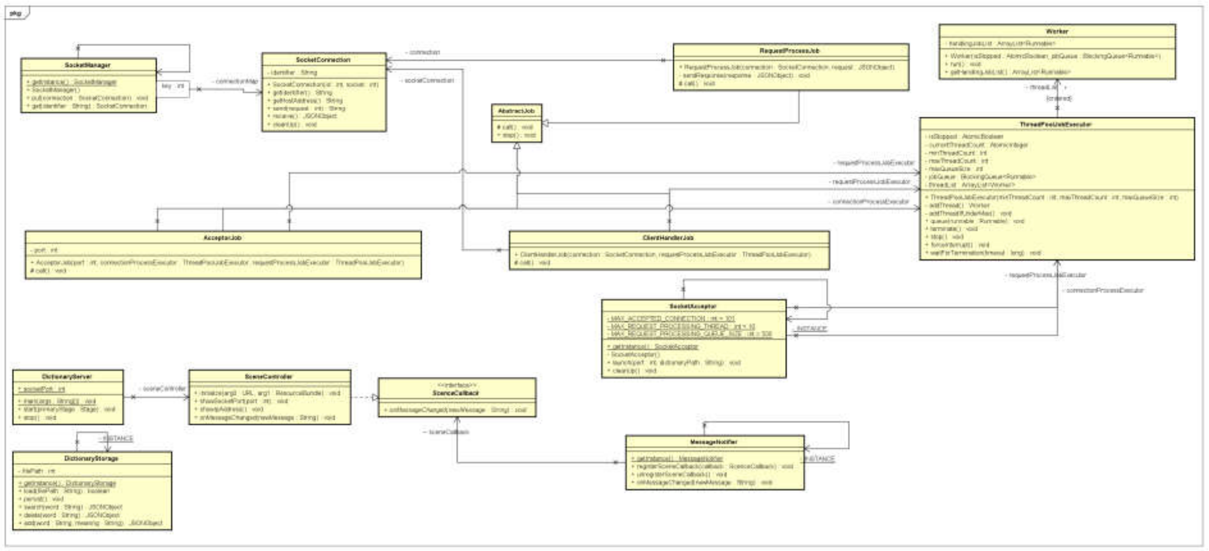


Figure 6 The Server Dictionary class diagram

4.2 Dictionary Client

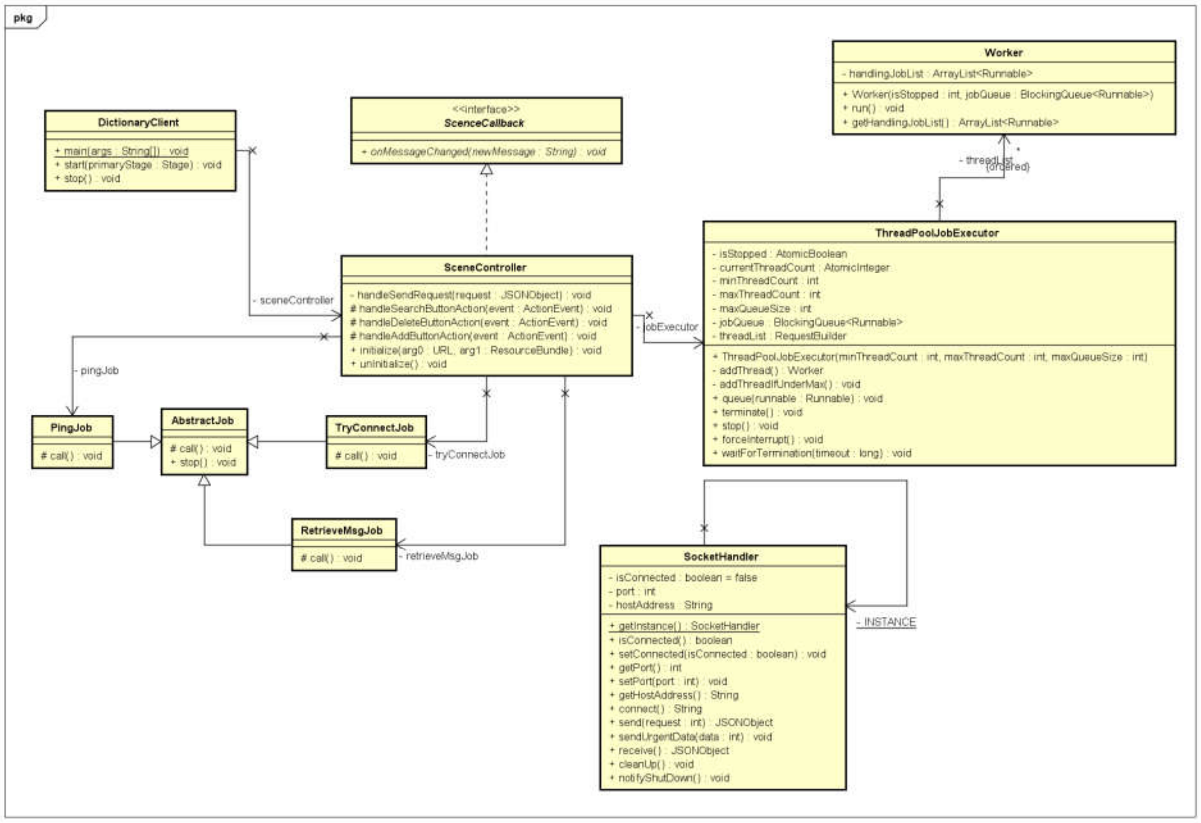

pkg
DictionaryClient
+ main(args : String[]) : void
+ start(primaryStage : Stage) : void
+ stop() : void
<<interface>>
ScenceCallback
+ onMessageChanged(newMessage : String) : void
SceneController
- handleSendRequest(request : JSONObject) : void
# handleSearchButtonAction(event : ActionEvent) : void
# handleDeleteButtonAction(event : ActionEvent) : void
# handleAddButtonAction(event : ActionEvent) : void
+ initialize(arg0 : URL, arg1 : ResourceBundle) : void
+ uninitialize() : void
- sceneController
- pingJob
- jobExecutor
- tryConnectJob
- retrieveMsgJob
PingJob
# call() : void
AbstractJob
# call() : void
+ stop() : void
TryConnectJob
# call() : void
RetrieveMsgJob
# call() : void
Worker
- handlingJobList : ArrayList<Runnable>
+ Worker(isStopped : int, jobQueue : BlockingQueue<Runnable>)
+ run() : void
+ getHandlingJobList() : ArrayList<Runnable>
- threadList
*
{ordered}
ThreadPoolJobExecutor
- isStopped : AtomicBoolean
- currentThreadCount : AtomicInteger
- minThreadCount : int
- maxThreadCount : int
- maxQueueSize : int
- jobQueue : BlockingQueue<Runnable>
- threadList : RequestBuilder
+ ThreadPoolJobExecutor(minThreadCount : int, maxThreadCount : int, maxQueueSize : int)
- addThread() : Worker
- addThreadIfUnderMax() : void
+ queue(runnable : Runnable) : void
+ terminate() : void
+ stop() : void
+ forceInterrupt() : void
+ waitForTermination(timeout : long) : void
SocketHandler
- isConnected : boolean = false
- port : int
- hostAddress : String
+ getInstance() : SocketHandler
+ isConnected() : boolean
+ setConnected(isConnected : boolean) : void
+ getPort() : int
+ setPort(port : int) : void
+ getHostAddress() : String
+ connect() : String
+ send(request : int) : JSONObject
+ sendUrgentData(data : int) : void
+ receive() : JSONObject
+ cleanUp() : void
+ notifyShutDown() : void
- INSTANCE

Figure 7 The Client Dictionary class diagram

5 Excellence

- The system is carefully designed and coded for readable, extensible, and maintainable abilities.
- Separate user interface design to fxml file. The application will load this file to render GUI when it starts up.
- Application is built on JavaFx so that it is easy to create, manage and customize the user interface.
- All processing logs and runtime errors are displayed in the user interface so that user can check the current workflow or troubleshooting when there are any errors during the application lifecycle.
- Only allow users to input letter characters in text field used to perform a search, delete or add operations. If users violate this constraint, a message dialog will be shown to inform this violation.
- Only allow users specifies socket port between 1024 and 49151 which are allocated for user ports according to IANA (Internet Assigned Numbers Authority).

6 Creativity

- A user interface is introduced for Dictionary Server application to display IP address and port using to establish a socket connection from client-side. Furthermore, all processing logs between Server and Clients are displayed on GUI so that the user can track the workflow as well as run-time errors.

- Design a thread pool job executor to manage threads and processing jobs rather than using Java's built-in thread pool.
- All client requests are queued and dispatched to allocated threads to concurrently process requests. Therefore, Sever GUI application is not freeze even if many Client applications are concurrently sending the request to Server.
- Do not need to launch Server application before launching Client application. The Client application will automatically establish the socket connection when Server is ready.
- The Client application will automatically re-establish the socket connection to Server when the user closes Server application or Server application is interrupted.
- The Client application will send a notification message to Server application when it is shutting down so that Server application has a chance to close corresponding socket connection and clean up all allocated resources for this Client.

7 Limitation

- The dictionary file is kept in memory during the Server application lifecycle. Only persist in-memory data to physical file when the Sever application shuts down. As a result, the modified content is not reflected to file if the Server application is interrupted.
- If the Client application cannot send shutdown notification to the Sever application due to interruption, the allocated resources such as the socket connection, input stream and output stream for Client is not cleaned up.

8 Deployment

- The release package includes:
  - DictionaryServer.jar
  - DictonaryClient.jar
  - Dictionary.json
- Make sure that JDK 12 is installed in the system and JavaFx 11 libraries folder is configured in the system variable in JAVA_FX_PATH name.
- Open terminal and run the following commands:
  - **Server**: java –jar --module-path %JAVA_FX_PATH% --add-modules javafx.controls,javafx.fxml DictionaryServer.jar 3005 "{YOUR PATH}\Dictionary.json"
  - **Client**: java –jar --module-path %JAVA_FX_PATH% --add-modules javafx.controls,javafx.fxml DictionaryClient.jar localhost 3005

9 Conclusion

A solution is provided to demonstrate knowledge about distributed system aspects:

- Client-Server architecture.
- Socket programming through TCP protocol.
- Threading and synchronization.
- User interface programming.

10 Reference

Buyya, R., Selvi, S.T. and Chu, X., 2009. Object-oriented Programming with Java: Essentials and Applications. Tata McGraw-Hill.