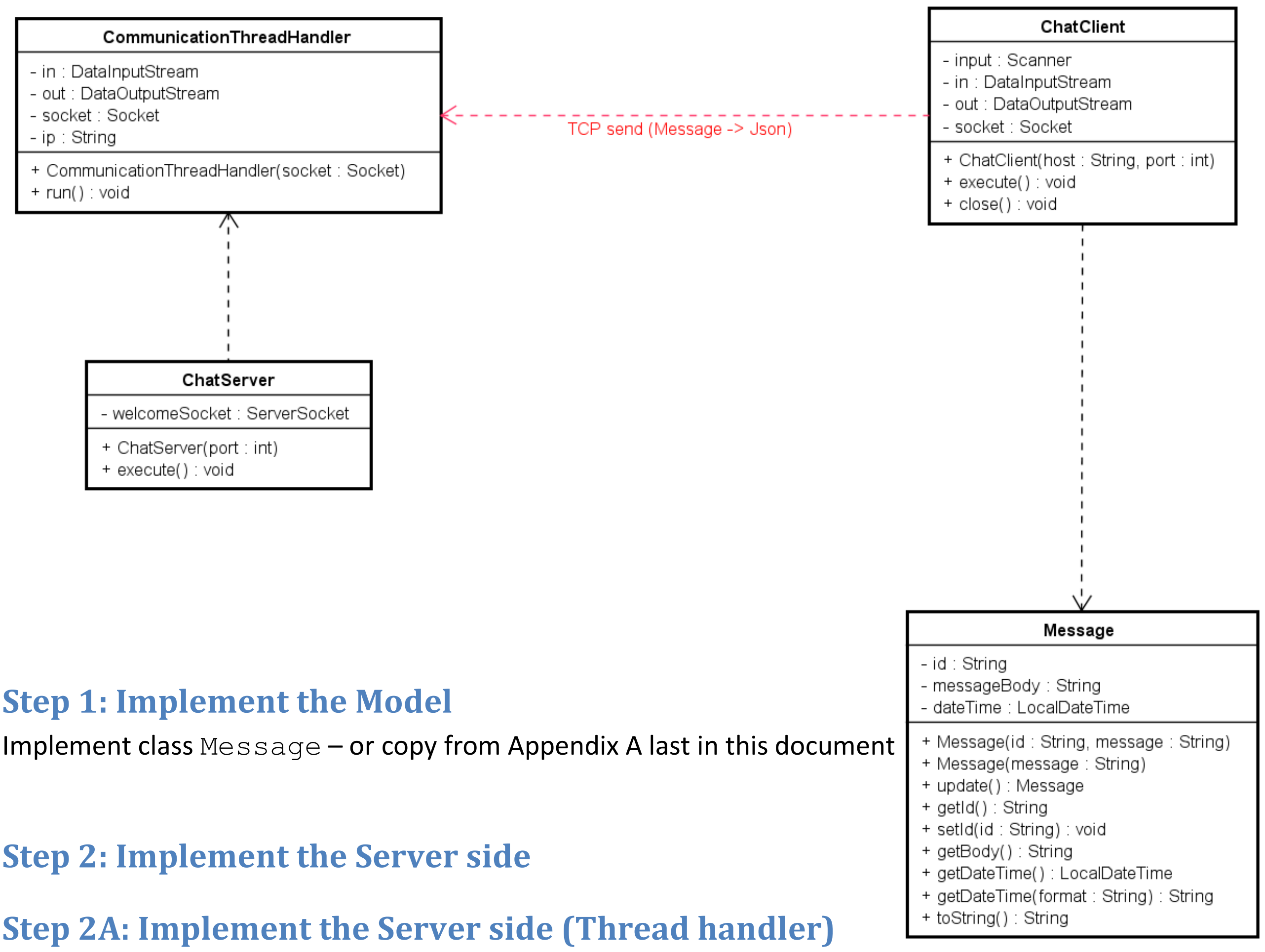
Exercises Session 6 SDJ2

Exercise 06.01 – A “chat” where clients send messages to the server



Step 1: Implement the Model

Implement class `Message` – or copy from Appendix A last in this document

Step 2: Implement the Server side

Step 2A: Implement the Server side (Thread handler)

Implement class `CommunicationThreadHandler`.

a) implementing `Runnable`
b) The constructor is initializing instance variables
c) Method `run` with a loop reading a `Json` string from the client, converting this to a `Message` object, and simply printing out the object. End the loop if the body of the message is “EXIT”.

Step 2B: Implement the Server side (ChatServer)

Implement class `ChatServer`.

a) The constructor is initializing instance variables
b) Method `execute` creates an infinite loop in which a client socket is created (`ServerSocket` method `accept()`) and a thread (with a `CommunicationThreadHandler` object) is created and started.

Step 2C: Implement the Server side (Server main)

Implement class `Server` with a main method, creating a `ChatServer` and calling `execute`.

Step 3: Implement the Client side

Step 3A: Implement the Client side (TaskListClient)

Implement class `ChatClient`.

a) The constructor is initializing instance variables

b) Method `execute` creates a loop in which you repeatedly

    1) Read an input text from keyboard

    2) Create a `Message` object with the input text as the message body

    3) Convert the `Message` object to a `Json` string

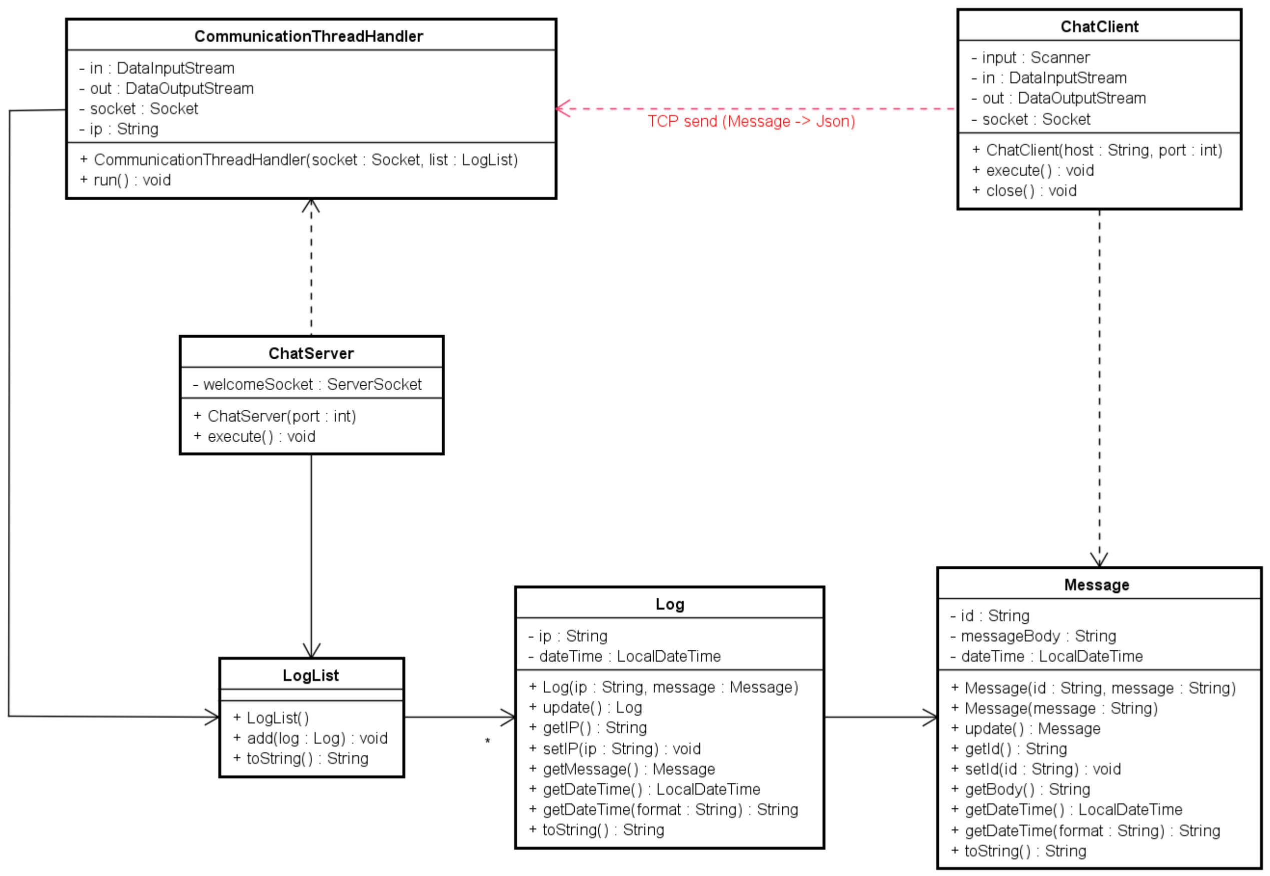
    4) Send the `Json` string to the server

c) Method `close` closes the socket and the keyboard stream (Scanner object)

Step 3B: Implement the Client side (Client main)

Implement class `Client` with a main method, creating a `ChatClient` and calling `execute`.

Exercise 06.02 – Logging messages on server side



Change the previous exercise such that you log the messages in a Log list. When you receive a `Message` object, you create a `Log` object with the client `IP` and the `Message` object and add this to the `LogList`. Classes `Log` and `LogList` are given in appendices. Remember to change the constructor in `CommunicationThreadHandler`.

Appendix A: Class Message

```java
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Message
{
   private String id;
   private String messageBody;
   private LocalDateTime dateTime;

   public Message(String id, String message)
   {
      this.dateTime = LocalDateTime.now();
      this.id = id;
      this.messageBody = message;
   }

   public Message(String message)
   {
      this("0", message);
      setId("" + (int) (messageBody.hashCode() * Math.random()));
   }

   public Message update()
   {
      this.dateTime = LocalDateTime.now();
      return this;
   }

   public String getId()
   {
      return id;
   }

   public void setId(String id)
   {
      this.id = id;
   }

   public String getBody()
   {
      return messageBody;
   }

   public LocalDateTime getDateTime()
   {
      return dateTime;
   }

   public String getDateTime(String format)
   {
      DateTimeFormatter formatter = DateTimeFormatter.ofPattern(format);
      return dateTime.format(formatter);
   }

   public String toString()
   {
      DateTimeFormatter formatter
               = DateTimeFormatter.ofPattern("d/MM/yyyy HH:mm:ss");
      return "id=" + id + ", time=" + dateTime.format(formatter)
            + ", message=\"" + messageBody + "\"";
   }
}
```

Appendix B: Class Log

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Log
{
   private String ip;
   private Message message;
   private LocalDateTime dateTime;

   public Log(String ip, Message message)
   {
      this.dateTime = LocalDateTime.now();
      this.ip = ip;
      this.message = message;
   }

   public Log update()
   {
      this.dateTime = LocalDateTime.now();
      return this;
   }

   public String getIP()
   {
      return ip;
   }

   public void setIP(String ip)
   {
      this.ip = ip;
   }

   public Message getMessage()
   {
      return message;
   }

   public LocalDateTime getDateTime()
   {
      return dateTime;
   }

   public String getDateTime(String format)
   {
      DateTimeFormatter formatter = DateTimeFormatter.ofPattern(format);
      return dateTime.format(formatter);
   }

   public String toString()
   {
      DateTimeFormatter formatter
          = DateTimeFormatter.ofPattern("d/MM/yyyy HH:mm:ss");
      return "IP=" + ip + ", time=" + dateTime.format(formatter)
            + ", message=[" + message + "]";
   }
}
```

Appendix C: Class LogList

```
import java.util.ArrayList;

public class LogList
{
   private ArrayList<Log> logs;

   public LogList()
   {
      logs = new ArrayList<>();
   }

   public void add(Log log)
   {
      logs.add(log);
   }

   public String toString()
   {
      return "" + logs;
   }
}
```