Hemuppgift 1: Sockets

Nätverksprogrammering, ID1212

Evan Saboo
saboo@kth.se

2017-11-17

1 Introduktion

Målet med första hemuppgiften är att kunna utveckla en applikation som kan kommunicera med en server över TCP eller UDP sockets, medans serven ska kunna hantera flera kommunikationer samtidigt.
Applikationen ska också ha en strukturerad arkitektur för att kunna dela upp logik och användargränssnitt, vilket leder till bättre programmeringsstruktur för framtidssäker utveckling.
Till sist ska applikationen kunna vara användarvänlig, dvs. den ska använda flera trådar för att göra användargränssnittet responsiv och för att dölja kommunikationsfördröjningar.

Uppgiften går ut på att man ska utveckla en klient-serverapplikation i Java för spelet ”Hangman”. Serven ska skicka ett gömt ord till spelaren och hen ska gissa bokstäverna i ordet eller gissa hela ordet. Serven ska ta hand om spelarens gissningar, dvs. det är serven som kollar om bokstaven/ordet är korrekt. Klienten ska inte spara någon data eller göra någon uträkning, det är servens jobb.

2 Litteraturstudie

Uppgiften utfördes med hjälp av dessa källor:

* Jag fick mest hjälp från Introduktions- och Sockets/Streams föreläsningarna. Koden jag har skrivit har fått mest inspiration från chatt exemplet i Sockets föreläsningen.

* Jag använde mig av StackOverflow hemsidan för att få hjälp med problem som uppstod och för att få kort och enkel information om några java syntaxer.[1]

* Java dokumentationshemsidan var också hjälpsam för att få Java syntaxförklaringar.[2]

[1] https://stackoverflow.com
[2] https://docs.oracle.com/javase/7/docs/api/

3 Metod

Här förklaras alla steg jag tog för komma till slutprodukten:

Jag började först med att kolla igenom chatt exemplaret i kursen för att förstå hur sockets fungerade och hur man ska sätta upp serven och klienten för att de ska kunna kommunicera med varandra. I början var det svårt att förstå hur koden fungerade eftersom jag inte var kunnig med att använda trådar i Java språket, men jag har använt trådar tidigare i andra språk. Jag behövde också repetera hur MVC (Model View Controller) arkitekturen fungerade. Det sista som behövde gås igenom var hur man använde sockets för att skicka meddelanden över TCP-anslutning, vilket inte var så svårt eftersom man behövde bara göra en enkel koppling mellan serven och klienten för att kunna skicka meddelanden.
Efter flera genomgångar förstod jag hur chattprogrammet fungerade och då visste jag hur jag skulle utveckla hangman spelet.

Utvecklingen gick mest åt att försöka implementera flera delar av chattprogrammet i min applikation. Det kändes lite som att jag bara tog koden från chattprogrammet men jag försökte ändå utveckla min egen kod, vilket var ganska svårt eftersom programmet hade flera likheter med min applikation än skillnader. Jag tyckte också att chattprogrammet hade bra arkitektur som man kunde följa.

Det var också viktigt för mig att ha ett lätt och begripligt användargränssnitt. Jag lät mina vänner testa programmet några gånger för att ge mig feedback om hur användarupplevelsen kunde förbättras. Jag tänkte från början använda Javafx för designen men då skulle jag inte ha mycket tid åt att finslipa alla små fel som kunde komma upp, så jag gick istället med kommando baserat UI.

Applikationen utvecklades i NetBeans IDE eftersom den har bra Java integration. Den har också bra debugging som jag använde när jag inte kunde hitta felet direkt.

4 Resultat

Applikationen har delats i två delar, klienten och serven. I Figur 1 kan man se hur programmet är uppdelat.
Klienten följer MVC arkitekturen, där paketet view innehåller kod för användargränssnittet, net innehåller logik för att kommunicera med serven och controller hanterar dataflödet mellan net och view.
Servern har också MVC mönster men den är lite annorlunda från klienten. Den har istället paketet net som frontend och model som hanterar alla logik och data. I net paketet finns all kommunikationslogik för att kunna kommunicera med klienten och i model finns logik för speluträkningar. Serven har också en controller som har samma uppgifter som klientens controller.
Två till paket finns i programmet, startup och common. Startup startar klienten och common innehåller variabler som både servern och klienten använder.
Koden för hela applikationen finns i Github.[3]

Nu kommer jag gå in i mer detalj hur klienten och serven är uppbyggda och hur de fungerar.

Klienten:
Klientens användargränssnitt är byggt på konsolinmatning för att ta in användarens inmatningar. I Figur 2 kan man se att användaren har tre val. Valet ”start” meddelar servern om att man vill börja spela en ny runda. Valet ”quit” meddelar servern att användaren kommer bryta kommunikationen, vilket görs i klientens net paket. Det sista valet ”guess” (default case) tar emot en bokstav eller ord från användaren för att skicka vidare serven. För att kontrollera om användaren har matat in rätt kommand så kollar jag om den tillhör enum kollektionen som finns i common paketet.
Innan användaren kan starta spelet måste klienten kopplas till serven, vilket görs i start metoden. Connect metoden tar emot ipadress, portnummer och nytt objekt (hanterar servermeddelanden) och skickar dem vidare till controller klassen (Figur 3). Connect metoden i controller använder CompletableFuture APIn för att skapa en ny tråd med hjälp av ForkJoinPool.comonPool() metoden som låter tråden utföra kopplingen till serven. Metoden sendToServer använder också CompletableFuture för att skicka meddelanden till serven. Genom att skapa en ny tråd för at hantera

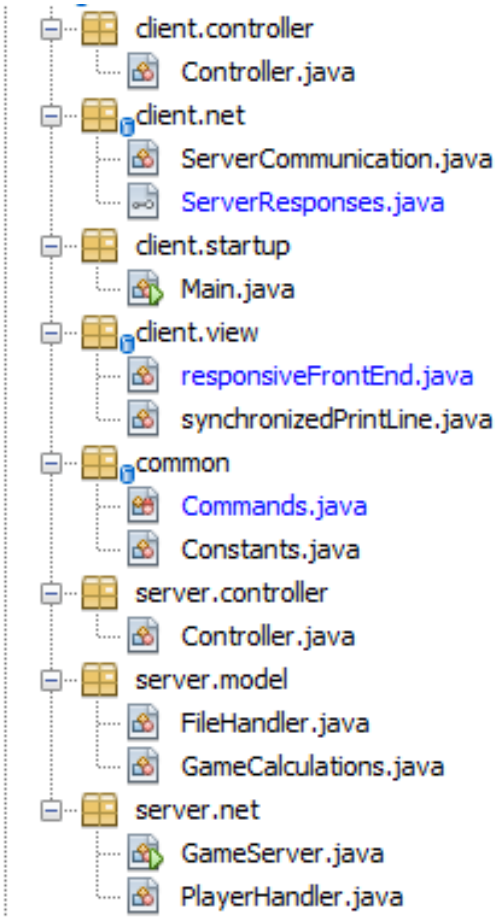
[3] https://github.com/ZpeedX/Network-Programming-Homework-1

uppgifter så kan man låta ”huvud tråden” hantera flera användarinmatningar, vilket gör att användargränssnittet blir responsiv och då kan användaren tex. avsluta programmet när hen vill.

Nu ska vi kolla hur klienten kommunicerar med serven (Figur 4).
Jag använde TCP sockets för att koppla klienten till serven. I connect metoden skapas en ny socket och sätter des IP adress och port nummer för att koppla till serven. Jag definierade också hur länge klienten ska försöka koppla till serven och hur läge kopplingen mellan dem ska vara aktiv medans ingen kommunikation sker. Två variabler skapades för att kunna kommunicera med serven, en som använder Socket.getOutputStream metod för att skicka meddelande till servern och en annan som använder Socket.getInputStream metod för att ta emot meddelanden från serven. Det sista som görs i connect metoden är att skapa ny tråd som tar emot alla meddelanden från serven hela tiden (Figur 5). I Figur 4 kan man se en metod som stänger kopplingen från serven och en annan metod som skickar meddelanden till serven.

Servern:
Jämfört med klienten så har severn ett net paket istället för view paket eftersom serven kommunicerar bara med klienter, man kan tänka sig att kommunikationen är serverns användargränssnitt. I figur 6 startar serven genom att lyssna på en specifik port nummer med hjälp sockets. Om en klient kopplar till serven så kommer den låta en annan tråd hantera kommunikationen med klienten för att kunna ta emot andra klienter.
I figur 7 hanteras klienten genom att utföra operationer beroende på vad klienten har skickat. Om spelaren vill starta ny runda kommer serven att skicka tillbaka ett nytt gömt ord till klienten vilket sedan visas till spelaren. Det nya ordet hämtas från en arraylist (Figur 9) som innehåller alla ord tagna från en fil.
Serven kan också ta emot gissningar från klienten, då kontrollerar serven gissningen genom att använda metoder från controller (Figur 8). Controller skickar vidare gissningen till model för att kontrollera om gissningen är korrekt eller inte (Figur 10). Det sista kommandot ”quit” stänger kopplingen mellan serven och klienten genom att använda en metod i Socket objektet, sedan slutar klient tråden i servern köras.
I GameCalculations klassen (Figur 9) initialiseras arraylistan genom att använda Filehandler klassens metoder (Figur 11) för att hämta alla ord från den specificerade filen.
Alla klienttrådar i serven använder samma instans av controller för att inte behöva duplicera arraylistan.

4.1 MVC



Figur 1: Programmets struktur följer MVC arkitekturen

4.2 Klient kod:

```
public void start() {
        if (receivingCmds) {
            return;
        }
        contr = new Controller();
        contr.connect(ipAddress, Constants.NETWORK_PORT, new msgToPlayer());
        receivingCmds = true;
        new Thread(this).start();
    }
    @Override
    public void run() {

        while(receivingCmds){
            try{
                String [] cmdLine = readNextLine().split(" ");
                if(!checkArrayLength(cmdLine)){
                    wrongInputMsg();
                    continue;
                }
                Commands cmd = Commands.valueOf(cmdLine[0].toUpperCase());
                switch(cmd){
                    case START:
                        contr.sendToServer(cmdLine[0]);
                        break;
                    case QUIT:
                        receivingCmds = false;
                        contr.disconnect();
                        break;
                    default:
                        if(cmdLine.length < 2){
                            wrongInputMsg();
                            continue;
                        }
                        contr.sendToServer(cmdLine[Constants.STRING_TYPE_INDEX], cmdLine[Constants.STRING_BODY_INDEX]);
                        break;
                }
               } catch (Exception e){
                  wrongInputMsg();
                }
            }
        }
```

Figur 2: Kod som hanterar användarens inmatningar.

```java
public class Controller {
    private final ServerCommunication gameServer = new ServerCommunication();

    /**
     * @see ServerCommunication#connect(java.lang.String, int, client.net.ServerResponses)
     * Uses completable future to assign the connect() task to the ForkJoinPool class.
     * The ForkJoinPool assigns the task to a thread in the pool and the thread returns to the pool
     * when it's done.
     */
    public void connect(String host, int port, ServerResponses msgToPlayer){
        CompletableFuture.runAsync(() -> {
            try{
                gameServer.connect(host, port, msgToPlayer);
            } catch(IOException e){
                throw new UncheckedIOException(e);
            }
        }).thenRun(() -> msgToPlayer.msgHandler("Connected to " + host + ":" + port));
    }
    /**
     *@see ServerCommunication#disconnect()
     * @throws IOException
     */
    public void disconnect() throws IOException {
        gameServer.disconnect();
    }

    /**
     *@see ServerCommunication#sendToServer(java.lang.String...)
     */
    public void sendToServer(String... playerCmd) {
        CompletableFuture.runAsync(() -> gameServer.sendToServer(playerCmd));
    }
}
```

Figur 3: Klientens controller klass.

```java
public class ServerCommunication {
    private static final int TIMEOUT_HALF_HOUR = 1800000;
    private static final int TIMEOUT_HALF_MINUTE = 30000;
    private Socket socket;
    private PrintWriter toServer;
    private BufferedReader fromServer;
    private volatile boolean connected;

    /**
     * Creates a new socket which connects to the server via port and ip address.
     * Also defines variables which sends messages to server and receives messages from server
     * Creates a listener thread which handles all received message from the server.
     * @param host IP address
     * @param port port number
     * @param msgToPlayer Server responses
     * @throws IOException
     */
    public void connect(String host, int port, ServerResponses msgToPlayer) throws IOException {
        socket = new Socket();
        socket.connect(new InetSocketAddress(host, port), TIMEOUT_HALF_MINUTE);
        socket.setSoTimeout(TIMEOUT_HALF_HOUR);
        connected = true;
        boolean autoFlush = true;
        toServer = new PrintWriter(socket.getOutputStream(), autoFlush);
        fromServer = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        new Thread(new ServerListener(msgToPlayer)).start();
    }

    /**
     * Send a "disconnect" command to server and close socket connection
     * @throws IOException Socket failure
     */
    public void disconnect() throws IOException {
        sendToServer(Commands.QUIT.toString());
        socket.close();
        socket = null;
        connected = false;
    }

/**
 * Send message to server
 * @param parts takes one or several string parameters and joins them
 */
public void sendToServer(String... parts){
    StringJoiner joiner = new StringJoiner(Constants.STRING_DELIMETER);
    for(String part : parts) {
        joiner.add(part);
    }
    toServer.println(joiner.toString());
}
```

Figur 4: En klass i net paketet (model) som har metoder för att koppla till-, skicka till- eller koppla ifrån servern.

```
    /**
     * A thread which only handles messages from the server
     */
    private class ServerListener implements Runnable {
        private final ServerResponses serverResponses;
        private ServerListener(ServerResponses serverResponses){
            this.serverResponses = serverResponses;
        }

        @Override
        public void run(){
            try{
                for(;;){
                    serverResponses.msgHandler(fromServer.readLine());
                }
            } catch(IOException e) {
                if(connected)
                    serverResponses.msgHandler("Connection lost");
            }
        }

    }
}
```

Figur 5: En klass i net paketet som tar emot serverns meddelande och skriver ut den till användaren.

4.3 Server kod

```
public GameServer(){
    contr = new Controller();
}
public static void main(String[] args){
    GameServer server = new GameServer();
    server.startServer();
}


/**
 * In independent thread which listens on the provided network port and
 * accepts new connections and pass them on to other new threads.
 */
private void startServer(){
    try{
        ServerSocket newPlayers  = new ServerSocket(Constants.NETWORK_PORT);
        while(true){
            Socket playerSocket = newPlayers.accept();
            playerHandler(playerSocket);
        }
    } catch(IOException e){
        System.err.println("Connection failed.");
    }
}

/**
 * Sets linger time and timout for the newly established socket and
 * it gets past to a new thread which handles all the game logic
 * @param playerSocket newly established socket to handle one player
 * @throws SocketException Socket failure
 */
private void playerHandler(Socket playerSocket) throws SocketException {
    playerSocket.setSoLinger(true, LINGER_TIME);
    playerSocket.setSoTimeout(TIMEOUT_HALF_HOUR);
    PlayerHandler handler = new PlayerHandler(playerSocket, contr);

    Thread playerThread = new Thread(handler);
    playerThread.setPriority(Thread.MAX_PRIORITY);
    playerThread.start();

}
```

Figur 6: Servens front end för att ta emot nya klienter. Varje ny klient tills en tråd som ska ta hand om klientens operationer.

```
    boolean autoFlush = true;
    fromPlayer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
    toPlayer = new PrintWriter(clientSocket.getOutputStream(), autoFlush);
    gameStarted = false;
    sendToPlayer("\nWelcome to the hangerman game!\n"
                + "1. To start a new game type 'start' \n"
                + "2. To guess a letter or word type the command 'guess' followed by a letter/
                + "3. To quit the game type 'quit'\n"
                + "Good Luck!");

} catch (IOException ioe) {
    throw new UncheckedIOException(ioe);
}

while (connected) {
    try {
        ReceivedMsg newMsg = new ReceivedMsg(fromPlayer.readLine());
        switch(newMsg.cmdType){
            case START:
                if(gameStarted){
                    sendToPlayer("You can't preform this action!");
                    continue;
                }
                startNewGame();
                break;
            case QUIT:
                closeConnection();
                break;
            case GUESS:
                if(!gameStarted){
                    sendToPlayer("Incorrect operation. You haven't started a game yet.");
                    continue;
                }
                guess(newMsg.stringBody);
                break;
            default:
                System.err.println("The command '"+ newMsg.receivedString+ "' is corrupt.");

        }
    } catch (IOException ioe) {
        System.out.println("Something went wrong, disconnecting client...");
        closeConnection();
    }

}
```

Figur 7: PlayerHandler klasser som hanterar spelarens operationer som kommer från klient sidan.

```java
public class Controller {
    private final GameCalculations model = new GameCalculations();

    /**
     * @see GameCalculations#getWord(java.util.ArrayList)
     * @param prevWords contains already used words
     * @return new word
     */
    public String getWord(ArrayList<String> prevWords){
        return model.getWord(prevWords);
    }

    /**
     * @see GameCalculations#guessLetter(java.lang.String, java.lang.String, char)
     * @return New hidden word or empty string
     */
    public String guessLetter(String word, String hiddenWord, char Letter){
        return model.guessLetter(word, hiddenWord, Letter);
    }

    /**
     * @see GameCalculations#guessWord(java.lang.String, java.lang.String)
     * @return if the two string match (boolean value)
     */
    public boolean guessWord(String word, String guessedWord){
        return model.guessWord(word, guessedWord);
    }

    /**
     * @see GameCalculations#hideWord(java.lang.String)
     * @return String containing only underscores
     */
    public String hideWord(String word){
        return model.hideWord(word);
    }

    /**
     * @see GameCalculations#removeSpaces(java.lang.String)
     * @param s string containing spaces
     * @return string without spaces
     */
    public String removeSpaces(String s){
       return model.removeSpaces(s);
    }
```

Figur 8: Controller klassen som hämtar metoder från model och skickar vidare till view

```java
public class GameCalculations {
    private ArrayList<String> words = new ArrayList();

    public GameCalculations(){
        try{
            words = toArrayList("resources/words.txt");
        } catch(IOException e){
            e.printStackTrace();
        }
    }

    /**
     * Uses readText to get all words from a file and inserts them to an arraylist.
     * @param filepath location of file
     * @return arraylist containing all words from a file
     */
    private ArrayList<String> toArrayList(String filepath) throws IOException{
        FileHandler fileHandler = new FileHandler();
        String file = fileHandler.readText(filepath);
        return new ArrayList<>(Arrays.asList(file.split(" ")));
    }

    /**
     * Gets an unused random word from ArrayList<String> words.
     * @param prevWords contains previous used words.
     * @return the random word
     */
    public String getWord(ArrayList<String> prevWords){
        Random rn = new Random();
        String word;
        while(true){
            int n = rn.nextInt(words.size()) +1;
            word = words.get(n);
            if(!prevWords.contains(word))
                return word.toLowerCase();
        }
    }
    /**
     * Remove all blank spaces from a string
     * @param word a String containing blank spaces
     * @return string without blank spaces
     */
```

Figur 9: GameCalculations klassen hanterar all logik. Klassen finns i model paketet.

```java
/**
 * Checks if the given letter is included in the given word.
 * If it's true add the letter to the "hidden word", in other ords replace character "_" in the hidden word with the letter.
 * @param word An english word
 * @param hiddenWord Contains letters and underscores
 * @param letter The given letter
 * @return hiddenWord containing the letter, else return empty string
 */
public String guessLetter(String word, String hiddenWord, char letter){

    String newWord = "";
    String testHWord = hiddenWord;

    hiddenWord = removeSpaces(hiddenWord);
    if(testHWord.contains(letter+ ""))
        return testHWord;

    else if(word.indexOf(letter) > -1){
        for(int i = 0; i < word.length(); i++){
            if(word.charAt(i) == letter){
                newWord += letter;
            }
            else
                newWord += hiddenWord.charAt(i);
            if(i < (word.length()-1))
                newWord += " ";
        }
        return newWord;
    } else
        return "";
}

/**
 * Check if two given strings match
 * @param word First string
 * @param guessedWord Second string
 * @return boolean value
 */
public boolean guessWord(String word, String guessedWord){
    return word.equals(guessedWord);
}

    /**
     * Replace all letter in the given string with underscore
     * @param word string contaning english letters
     * @return String contaning which only contains underscores
     */
    public String hideWord(String word){
        return word.replaceAll(".", "_ ");
    }
}
```

Figur 10: Andra metoder i GameCalculations klassen.

```java
public class FileHandler {
    private static final String BLANK_SPACE = " ";

    /**
     * Takes a filepath to file in the filesystem and converts the content of the file to string
     * @param filePath path to a file
     * @return file content
     * @throws IOException file not found or error reading file
     */
    public String readText(String filePath) throws IOException{
        try(BufferedReader fileContent = new BufferedReader(new FileReader(filePath))) {
            StringBuilder textBasedContent = new StringBuilder();
            fileContent.lines().forEachOrdered(line -> appendElement(textBasedContent, line));
            return stringBuilderToString(textBasedContent);
        }
    }

    /**
     * Adds a line to stringbuilder a blank space after the line
     * @param sb Stringbuilder
     * @param line string
     */
    private void appendElement(StringBuilder sb, String line) {
        sb.append(line);
        sb.append(BLANK_SPACE);
    }

    /**
     * Converts stringbuilder to string
     * @param sb Stringbuilder
     * @return String
     */
    private String stringBuilderToString(StringBuilder sb) {
        return sb.toString().trim();
    }
}
```

Figur 11: Klassen FileHandler är till för att hämta nödvändiga filer från Windows filsystem. Klassen finns också i model paketet.

5 Diskussion

Kraven nedan uppfylldes:
*Programmet följer MVC arkitekturen för att ha bättre struktur på logiken och användargränssnittet och kunna kontrollera alla operationer som används mellan dem.
*Jag använde TCP-sockets för att serven och spelaren ska kunna kommunicera med varandra under spelets gång.
*Serven tog hand om all data och logik för kunna hantera spelarens inmatningar.
* För att spelaren ska ha en reponsiv användargränssnitt behövde jag använda flera trådar som tog hand användarimatningar och server kommunikationen.
*Serven behövde också använda flera trådar för att kunna hantera flera klienter samtidigt.

De flesta av problemen som uppstod under utvecklingen var inte stora nog för att bli frustrerad över. Det största problemet som uppstod var när jag skulle läsa från en fil. När jag försökte öppna en fil så kunde inte programmet hitta filen. Jag försökte hitta flera lösningar till problemet genom att tex. låta programmet läsa från en annan plats eller kopiera filen till flera mappar men problemet var fortfarande olöst.
Efter en timme av debugging kom jag fram till att Netbeans körde gammal kod som producerade samma resultat hela tiden, vilket löstes genom att starta om Netbeans.

Jag är ganska nöjd med min applikation men om jag hade lite mer tid så skulle jag ha använt Javafx för bättre användarupplevelse.

6 Kommentar om kursen

Jag har spenderad 10–12 timmar åt att gå igenom föreläsningarna innan jag började med hemuppgiften. Tiden det tog att utföra hemuppgiften tog 15–20 timmar. Rapporten tog ungefär 5 timmar att skriva.