Learning Journal Unit\_7

Dawid Blom

**Thursday, 20/05/2021, 8 – 12 am: Reading Assignment:** This week in the reading assignment, I read more information regarding the Swing framework. We went into more detail about the Model-View-Controller design and how it is implemented. Things make a lot more sense After I read this chapter, and it gave me sense of I am well rounded in how to go about implementing GUI applications in Swing. However, I do believe this same pattern can be applied to any form of software being developed. Therefore, I think I should study this area more in depth. Furthermore, I read about web browsers and how to go about implementing one. I cannot wait for the lab since we will actually be implementing one of these. I feel that this really rounded things off and it is in fact coming to an end. Nonetheless, I learned a lot from this chapter and it really helped me to better understand the previous chapters we did as well.

**Thursday, 20/05/2021, 1 – 2 pm: Self-Quiz:** The self quiz was fun, It gave a lot more code examples that we have to figure out and understand in order to be able to give an answer. This truly helps me build a more programmer mind set and how to think code. Furthermore, It also helped me grasp certain areas of the reading material we had to do this week.

**Thursday, 20/05/2021, 2 – 3 pm: Discussion Assignment:** The discussion post for this week handled the MVC (Model-View-Controller), and it really helped me to get more in depth knowledge regarding this subject. In all honesty, I believe this subject is a massive one for us student to get a more big picture grasp of how to go about designing software with a pattern that has with stood test of time.

**Sunday, 23/05/2021, 8 – 2 pm: Programming Assignment:** This week’s programming assignment was fun and interesting since it had a lot to do with network programming. However, I felt that the assignment was way out of scope of what we have been doing, considering that the assignment had a --multi threading-- part to it. Therefore, I felt lost and very confused. Nonetheless, it forced me to go do some research on my own and learn the basics of multi threading.

import java.net.ServerSocket;

import java.net.Socket;

import java.util.Scanner;

/\*\*

\* The main() program in this class is designed to read requests from

\* a Web browser and display the requests on standard output. The

\* program sets up a listener on port 50505. It can be contacted

\* by a Web browser running on the same machine using a URL of the

\* form http://localhost:505050/path/to/resource.html This method

\* does not return any data to the web browser. It simply reads the

\* request, writes it to standard output, and then closes the connection.

\* The program continues to run, and the server continues to listen

\* for new connections, until the program is terminated (by clicking the

\* red "stop" square in Eclipse or by Control-C on the command line).

\*/

public class ReadRequest {

/\*\*

\* The server listens on this port. Note that the port number must

\* be greater than 1024 and lest than 65535.

\*/

private final static int LISTENING\_PORT = 50505;

/\*\*

\* Main program opens a server socket and listens for connection

\* requests. It calls the handleConnection() method to respond

\* to connection requests. The program runs in an infinite loop,

\* unless an error occurs.

\* @param args ignored

\*/

public static void main(String[] args) {

ServerSocket serverSocket;

try {

serverSocket = new ServerSocket(LISTENING\_PORT);

}

catch (Exception e) {

System.out.println("Failed to create listening socket.");

return;

}

System.out.println("Listening on port " + LISTENING\_PORT);

try {

while (true) {

Socket connection = serverSocket.accept();

System.out.println("\nConnection from "

+ connection.getRemoteSocketAddress());

handleConnection(connection);

}

}

catch (Exception e) {

System.out.println("Server socket shut down unexpectedly!");

System.out.println("Error: " + e);

System.out.println("Exiting.");

}

}

/\*\*

\* Handle commuincation with one client connection. This method reads

\* lines of text from the client and prints them to standard output.

\* It continues to read until the client closes the connection or

\* until an error occurs or until a blank line is read. In a connection

\* from a Web browser, the first blank line marks the end of the request.

\* This method can run indefinitely, waiting for the client to send a

\* blank line.

\* NOTE: This method does not throw any exceptions. Exceptions are

\* caught and handled in the method, so that they will not shut down

\* the server.

\* @param connection the connected socket that will be used to

\* communicate with the client.

\*/

private static void handleConnection(Socket connection) {

try {

Scanner in = new Scanner(connection.getInputStream());

while (true) {

if ( ! in.hasNextLine() )

break;

String line = in.nextLine();

if (line.trim().length() == 0)

break;

System.out.println(" " + line);

}

}

catch (Exception e) {

System.out.println("Error while communicating with client: " + e);

}

finally { // make SURE connection is closed before returning!

try {

connection.close();

}

catch (Exception e) {

}

System.out.println("Connection closed.");

}

}

}

**Sunday, 23/05/2021, 2:30 – 5 pm: Journal Assignments:** The StopWatch code we had to implement was very fun and interesting. I enjoyed very much since we had to use a few libraries we have not used in a while.

import java.awt.event.\*;

import javax.swing.\*;

/\*\*

\* A custom component that acts as a simple stop-watch. When the user clicks

\* on it, this componet starts timing. When the user clicks again,

\* it displays the time between the two clicks. Clicking a third time

\* starts another timer, etc. While it is timing, the label just

\* displays the message "Timing....".

\*/

public class StopWatchLabel extends JLabel implements MouseListener {

private long startTime; // Start time of timer.

// (Time is measured in milliseconds.)

private boolean running; // True when the timer is running.

/\*\*

\* Constructor sets initial text on the label to

\* "Click to start timer." and sets up a mouse listener

\* so the label can respond to clicks.

\*/

public StopWatchLabel() {

super(" Click to start timer. ", JLabel.CENTER);

addMouseListener(this);

}

/\*\*

\* Tells whether the timer is currently running.

\*/

public boolean isRunning() {

return running;

}

/\*\*

\* React when the user presses the mouse by starting

\* or stopping the timer and changing the text that

\* is shown on the label.

\*/

public void mousePressed(MouseEvent evt) {

if (running == false) {

// Record the time and start the timer.

running = true;

startTime = evt.getWhen(); // Time when mouse was clicked.

setText("Timing....");

}

else {

// Stop the timer. Compute the elapsed time since the

// timer was started and display it.

running = false;

long endTime = evt.getWhen();

double seconds = (endTime - startTime) / 1000.0;

setText("Time: " + seconds + " sec.");

}

}

public void mouseReleased(MouseEvent evt) { }

public void mouseClicked(MouseEvent evt) { }

public void mouseEntered(MouseEvent evt) { }

public void mouseExited(MouseEvent evt) { }

}