CS2133: Computer Science II
Assignment 5

Prof. Christopher Crick

1 Minesweeper revisited (30 points)

In this problem, we return to the Minesweeper game you wrote two assignments ago. First of all, if you did not get full credit on this question on Assignment 3, you have a chance to earn back some points here. Return to your code, now that we have gone over it and you have seen how to construct a working version using good MVC design. Refactor and fix it until you have a working game. This will net you half the points you lost on assignment 3.

For this assignment, you will enhance your Minesweeper game with a couple of features. Add a menu bar to your JFrame, with a "File" menu. This menu should include four menu items: "New", "Save", "Load" and "Quit".

- When a user selects "New", the game should elicit information about how difficult the new game should be. Exactly how this works is up to you – specifying a gameboard size, odds of encountering a mine, or simply selecting a difficulty level.
- "Save" should save the current game state to disk. Present the user with a file chooser dialog for specifying a name and place to save the game. Hint: If you have implemented the MVC architecture well, this should be only a couple of lines of code.
- "Load", by contrast, should load a saved game off the disk. Again, present the user with a file chooser. Handle the case where the user chooses a file that is not in fact a saved Minesweeper game. When the game has loaded, the board should look just like it did when the user saved it before.
- "Quit" should simply end the program. The program should *only* quit when the user selects this option or clicks the JFrame's close box. If the game is lost, the user should be able to look at the revealed game board and then select "New" to start again.

2 Your own webserver (35 points)

In Assignment 4, you wrote a simple web browser client. This time around, you will write a multithreaded webserver which will serve HTML files from the filesystem to a browser client. Write a program called Webserver.java that creates a ServerSocket object and then enters an infinite loop. The program should take the Socket returned by the ServerSocket's accept() method, create a Runnable object that uses the Socket, pass this object to a Thread, start the Thread running, and go back to waiting for another client connection.

The Runnable object will handle establishing the connection and serving the file. You already know what the client's handshake string looks like:

```
GET <filename> HTTP/1.1
```

Your server should receive this string and extract the name of the file. If the request is empty (ie the name of the file is simply /), then by convention the server should load a file called "index.html".

You will need to look for the requested file on the filesystem and load it into memory. You will then respond to the client's request using the simplest possible HTTP response:

```
HTTP/1.1 200 OK\r\n
Content-type: text/html\r\n\r\n
```

Backslash-r and backslash-n here correspond to the carriage return and newline characters, respectively. After you send these lines, you should send the text file.

If the browser asks for a file that does not exist, you should instead send the response

```
HTTP/1.1 404 Not Found\r\n\r\n
```

If anything else goes wrong you can try to send

```
HTTP/1.1 500 Internal Server Error\r\n\r\n
```

Important: This server is extremely insecure, and will allow a malicious user to access any textfile on your machine. You should implement a check so that the server will only send files that are located inside a particular directory designated by you. If the user tries to navigate outside of that directory (by using the parent-directory symbol ".."), the server should not permit it.

Extra credit: Allow the server to serve images as well as HTML files. Check the filename, and if it ends with ".jpg", change the HTTP content-type from "text/html" to "image/jpeg", and load and transmit as a binary file rather than a textfile.

3 Sorting and orders of growth (35 points)

Implement a program (command-line one will do) that generates two double arrays of length n, filled with identical random doubles between 0 and 1. Sort one array using bubble sort, and the other using merge sort (both of which you must implement). Time each execution and print the result. If it takes longer than 20 seconds, stop executing and report the fact. If you run out of memory, stop executing and report the fact. Exception handling will help here – don't let your program crash or run for hours.

Your program should do this over and over until you either run out of memory or neither sort finishes within twenty seconds. Start n out at ten, and multiply by 10 each time, so you're testing the algorithm at $n = 10, 100, 1000, 10000$, etc.

Write a couple of lines in the code's comments discussing the running time difference between $O(n^2)$ and $O(n \log n)$ sorting methods.

Turning in

The files with the main() function that actually execute the programs should be named Minesweeper.java, Webserver.java and Sorting.java, so we know which ones we're supposed to run. Those three, plus all of the other .java files that define the classes you need for these programs, should be wrapped up in a zip file called assignment_5_<your_name>.zip and uploaded to the Dropbox at oc.okstate.edu. Ensure that everything can be compiled and run from the command line. This assignment is due Wednesday, April 9, at noon.