HTTPServer / HTTPClient

Introduction.

This report covers the implementation of a guessing game in the web browser utilizing cookies for user identification. Furthermore, the java.net.HttpURLConnection class is used to simulate a browser and play the game 100 times and present the average number of guesses.

HTTPServer

In this class we create a server socket on port 8080 and accept every incoming request. We use a tokenizer to extract the different http header fields. We look for the *guess* variable in the request and if this exists we save the given value for later use. If the http header contains a cookie we save it for later use. If this cookie is a cookie we have created and given out in a previous response we do a printout statement that this is the case and scan for if the user wants to start a new game. If the cookie is not known we create a new one by using the UUID library and create a new game.

When the request has been handled we start the response. Create a print stream on the socket and send back the 200 OK message and info about our server. We also use the set-cookie field to give the new user its existing cookie or the newly created one. Next is the data we want to send to the client. We check if the guess given by the client is correct or not, this is then sent back by embedding the guess in a html code sequence.

Game

The Game class is responsible for handling the game logic. The first thing this class does is to create a simple Database class holding all the important data about the users. The initial idea was to instantiate a new Game object for each client and save them all in a hash map in the server class but this led to problems when using multiple browsers at the same time for an unknown reason. The solution to this was to create a simple database of two hash maps holding the data. This approach could maybe be more memory efficient since objects require a relative large amount of memory to use. Now we use the same Game object for all clients. However, The Game class consists of three simple methods. The most important is check() which handles the game logic and checks if the given guess from the user is correct or not. The method takes the guess and the cookie from the client and increment the number of guesses and in addition to this returns a simple html tag with the answer by looking in the databases hash map for the correct cookie and compares the secret number for the client with the given guess. The two other methods newGame() and existingClient() creates a new game and checks if the cookie given by the client already is in use and has an ongoing game.

Database

This class is responsible for storing clients cookies and associate them with their secret number as well as the number of guesses they have made. This is done in two hash maps.

The method existingClient() checks if there is an existing client with the given cookie in the key value store, it returns a boolean.

addNum() adds a secret number together with the clients cookie in the hash map and also saves the users cookie in the hash map that stores the number of guesses. Lastly, it prints the clients cookie and its corresponding secret number to standard output.

getNum() returns the secret number associated with the given cookie.

incrementGuess() increments the number of guesses the clients has made.

getGuesses() returns the number of guesses the clients has made.

All of these methods are called from the Game class, the class is invisible for the HTTPServer class.

Extra assignment - HttpURLConnection

This class simulates a browser and plays the game 100 times and presents the average number of guesses.

The class starts by getting a cookie by opening a connection to the url where our server and game is running. We use getHeaderFields() to get the cookie from the servers response and saves it for future requests.

A new game is then started where we increment the number of games played and open a new connection with setRequestProperty(”Cookie”, cookie) to identify ourselves. We then enter the makeGuess() method where we calculate what guess we should make. The guess is then passed to sendGuess() which is responsible for sending our guess in a http request to the server. The server will return some html code and the method scans this for keywords to determine if the guess was correct, too high or too low. This is then returned to the makeGuess() method so that it can optimize its next guess. If the guess was right it is printed to standard output and if the number of games played is lower than the specified number a new game is started and the steps are performed once again. Lastly, we calculate the average number of guesses by dividing the total number of guesses by the number of gasmes played.