\*\*\*\*\*\*\*\*\*\*\*Board.java\*\*\*\*\*\*\*\*\*\*\*\*\*8

package server;

import java.util.Arrays;

import java.util.Iterator;

import java.util.LinkedList;

import java.util.List;

import command.Command;

/\*\*

\* Object which represents a whiteboard stored on the server. Stores a list

\* of all commands ever sent to whiteboard so that it can be recreated on all

\* clients. Also stores all current users connected to this whiteboard.

\*

\* Concurrency Argument:

\* - This class is made concurrent by the monitor pattern

\*

\*/

public class Board {

private LinkedList<Command> commands = new LinkedList<Command>();

private List<String> users = new LinkedList<String>();

/\*\*

\* Deletes user from board if user is in board

\* @param username

\*/

public synchronized void deleteUser(String username) {

Iterator<String> it = users.iterator();

while(it.hasNext()) {

String user = it.next();

if (user.equals(username)) {

it.remove();

}

}

}

/\*\*

\* Adds user to board

\* @param username

\*/

public synchronized void addUser(String username) {

users.add(username);

}

/\*\*

\* Checks if username is available on this board

\* @param username

\* @return: true if username is not contained in board's users, false otherwise

\*/

public synchronized boolean checkUsernameAvailable(String username) {

return !users.contains(username);

}

/\*\*

\* Returns all commands ever sent to this board

\* @return

\*/

public synchronized LinkedList<Command> getCommands() {

return commands;

}

/\*\*

\* Adds a command to the board

\* @param command

\*/

public synchronized void addCommand(Command command) {

this.commands.add(command);

}

/\*\*

\* Returns list of all users in board

\* @return all users as an array

\*/

public synchronized String[] getUsers() {

String[] usersArray = new String[users.size()];

return users.toArray(usersArray);

}

/\*\*

\* Sets the users (shortcut for testing)

\* @param newUsers: the new array to set the users to

\*/

public synchronized void setUsers(String[] newUsers) {

LinkedList<String> usersList = new LinkedList<String>();

usersList.addAll(Arrays.asList(newUsers));

users = usersList;

}

}

\*\*\*\*\*\*\*\*\*\*\*Server.java\*\*\*\*\*\*\*\*\*\*\*

package server;

import java.io.IOException;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.Arrays;

import java.util.Hashtable;

import java.util.LinkedList;

import java.util.List;

import java.util.NoSuchElementException;

import java.util.Queue;

import command.Command;

/\*\*

\* Server for whiteboard application

\*

\* Concurrency Argument:

\* - All board objects are thread safe (see Board.java)

\* - All methods that modify this objects data representation are

\* made concurrent via the monitor pattern

\*/

public class Server {

//stores all the boards created as Board objects associated with names

private Hashtable<String, Board> boards = new Hashtable<String, Board>();

private List<Socket> clients = new LinkedList<Socket>();

private final ServerSocket serverSocket;

private boolean running;

/\*\*

\* Create our server on port port

\* @param port: port for server to listen on

\* @throws IOException

\*/

public Server(int port) throws IOException {

running = true;

serverSocket = new ServerSocket(port);

// Add shutdown hook to close server gracefully

addShutDownHook();

}

/\*\*

\* Run the server, listening for client connections and handling them.

\* Never returns unless an exception is thrown.

\*

\* @throws IOException if the main server socket is broken

\* Note: (IOExceptions from individual clients do \*not\* terminate serve())

\*/

public void serve() {

System.out.println("Server serving");

try {

// block until a client connects

while (running) {

Socket socket = serverSocket.accept();

clients.add(socket);

// create new thread for each connection

new Thread(new ServerProtocol(socket, this)).start();

}

} catch (IOException e) {

}

System.out.println("Server Shut down");

}

/\*\*

\* Add the command on the server's queue of commands Requires valid board

\* name

\*

\* @param boardName: the board to draw on

\* @param command: the command to perform on the board

\*/

public void updateBoard(String boardName, Command command) {

boards.get(boardName).addCommand(command);

}

/\*\*

\* Iterates through all the sockets and sends the command to each

\*

\* @param Command - command to be sent to all clients

\*/

public void sendCommandToClients(Command command, Socket skip) {

for (Socket client: clients) {

if (!skip.equals(client)) {

try {

if (!client.isClosed()) {

System.out.println("sending to client");

PrintWriter out = new PrintWriter(client.getOutputStream(), true);

out.println(command.toString());

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

/\*\*

\* Checks if the board name is unique

\* Creates a new board with the specified board name

\*

\* @param boardName: the name of the new board

\* @return: whether or not the new board was successfully made

\*/

public synchronized boolean newBoard(String boardName) {

if(boards.containsKey(boardName)) {

return false;

} else {

boards.put(boardName, new Board());

return true;

}

}

/\*\*

\* Removes the user from the old board and adds the user to the new board.

\*

\* @param username: the username of the user making the switch

\* @param oldBoardName: name of the board the user is switching from

\* @param newBoardName: the name of the board the user is switching to

\* @return: List of Commands of the new Board the user is switching to

\*/

public void switchBoard(String username, String oldBoardName, String newBoardName) {

boards.get(oldBoardName).deleteUser(username);

boards.get(newBoardName).addUser(username);

}

/\*\*

\* Gets the users from a board

\*

\* @param boardName

\* @return String which is a list of all users on the board

\*/

public String getUsers(String boardName) {

Board board = boards.get(boardName);

String[] users = board.getUsers();

StringBuilder usersString = new StringBuilder("");

for (String user: users) {

usersString.append(user + " ");

}

if(usersString.length() > 0) {

usersString.deleteCharAt(usersString.length() - 1);

}

return usersString.toString();

}

/\*\*

\* Gets a list of all the board names

\* @return: a String listing all of the board names with a space in front

\*/

public synchronized String getBoards() {

String[] boardsArray = boards.keySet().toArray(new String[0]);

StringBuilder boardsString = new StringBuilder("");

for (String board: boardsArray) {

boardsString.append(" "+board);

}

return boardsString.toString();

}

/\*\*

\* Checks if the username is unique and if it is, return true and enter the user

\* @param username: the username to check

\* @param boardName: the board the user wants to enter

\* @return: whether or not the user entered successfully

\*/

public synchronized boolean checkUser(String username, String boardName) {

for (String board : boards.keySet()) {

if (!boards.get(board).checkUsernameAvailable(username)) {

return false;

}

}

// If user is unique, add them to board

enter(username, boardName);

return true;

}

/\*\*

\* Adds the user to a board for the first time

\* @param username: the entering user

\* @param boardName: the board they have chosen to enter

\*/

public synchronized void enter(String username, String boardName) {

Board board = boards.get(boardName);

board.addUser(username);

}

/\*\*

\* Removes the user from all boards

\* @param username: the username of the user exiting

\*/

public synchronized void exit(String username) {

for(String boardName: boards.keySet()) {

Board board = boards.get(boardName);

board.deleteUser(username);

}

}

/\*\*

\* Gets all commands sent to a specific board

\* @param boardName

\* @return

\*/

public LinkedList<Command> getCommands(String boardName) {

return boards.get(boardName).getCommands();

}

/\*\*

\* Returns clients connected to server

\* @return

\*/

public List<Socket> getClients() {

return clients;

}

/\*\*

\* Get a certain board from the server

\* @param boardName: the name of the board to get

\* @return the board

\*/

public Board getBoard(String boardName) {

return boards.get(boardName);

}

/\*\*

\* Gets the hashtable of board names associated with boards

\* @return the hashtable

\*/

public Hashtable<String, Board> getBoardsHashtable() {

return boards;

}

/\*\*

\* Shuts down all client connections and then shuts down serverSocket

\* @throws IOException

\*/

public void shutDown() throws IOException {

running = false;

for (Socket client: clients) {

if (!client.isClosed()) client.close();

}

serverSocket.close();

}

public void close() throws IOException {

serverSocket.close();

}

public void addShutDownHook() {

// Add shutdown hook to shutdown server gracefully

Runtime.getRuntime().addShutdownHook(new Thread() {

public void run() {

try {

shutDown();

} catch (IOException e) {

e.printStackTrace();

}

}

});

}

/\*\*

\* Main method to launch server from command line

\* @param args

\*/

public static void main(String[] args) {

int port = 4444; // default port

// Check for and parse command line arguments

Queue<String> arguments = new LinkedList<String>(Arrays.asList(args));

try {

while ( ! arguments.isEmpty()) {

String flag = arguments.remove();

try {

if (flag.equals("--port")) {

port = Integer.parseInt(arguments.remove());

if (port < 0 || port > 65535) {

throw new IllegalArgumentException("port " + port + " out of range");

}

} else {

throw new IllegalArgumentException("unknown option: \"" + flag + "\"");

}

} catch (NoSuchElementException nsee) {

throw new IllegalArgumentException("missing argument for " + flag);

} catch (NumberFormatException nfe) {

throw new IllegalArgumentException("unable to parse number for " + flag);

}

}

} catch (IllegalArgumentException iae) {

System.err.println(iae.getMessage());

System.err.println("usage: Server [--port PORT]");

return;

}

// Try to launch the server

try {

Server server = new Server(4444);

server.serve();

} catch (IOException e) {

System.out.println("Error in starting server.");

e.printStackTrace();

}

}

}

\*\*\*\*\*\*\*\*\*ServerProtocol.java\*\*\*\*\*\*\*\*\*\*\*\*\*

package server;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

import java.util.List;

import command.Command;

/\*\*

\* Thread which handles each individual connection with each client and

\* communicates through the following grammar

\*

\* Concurrency Argument:

\* - this thread only performs actions on thread safe objects (Board, Server)

\* (See Board.java and Server.java)

\*

\*

\*/

public class ServerProtocol implements Runnable {

private final Socket socket;

private final Server server;

public ServerProtocol(Socket socket, Server server) {

this.socket = socket;

this.server = server;

}

/\*\*

\* Waits on the client to send data then calls the appropriate request handler

\*/

@Override

public void run() {

// handle the connection with the client

try {

handleConnection(socket);

} catch (IOException e) {

e.printStackTrace();

} finally {

try {

if (!socket.isClosed()) socket.close();

} catch (IOException e) {

}

}

}

/\*\*

\* Handle a single client connection. Returns when client disconnects.

\*

\* @param socket socket where the client is connected

\* @throws IOException if connection has an error or terminates unexpectedly

\*/

private void handleConnection(Socket socket) throws IOException {

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

try {

for (String line = in.readLine(); line != null; line = in.readLine()) {

try {

String output = handleRequest(line);

if(output != null) {

out.println(output);

}

} catch (IllegalArgumentException e) {

e.printStackTrace();

}

}

} finally {

out.close();

in.close();

}

}

/\*\*

\* Handler for client input, performing requested operations and returning an output message.

\*

\* Receives:

\*

\* New Board = "newBoard boardName"

\* Switch Board = "switch username oldBoardName newBoardName"

\* Exit = "exit username"

\* Draw = "draw boardName command param1 param2 param3 ... "

\* Example: "draw boardName drawLineSegment x1 y1 x2 y2 color width"

\* Get Users = "users boardName"

\* Get boards = "boards"

\* Check and add User = "checkAndAddUser username boardName"

\*

\*

\* Sends:

\*

\* New Board = "newBoard boardName boolean"

\* Switch Board = "switch username oldBoardName newBoardName command1 command2 command3..."

\* Update Users = "users boardName user1 user2 user3..."

\* Update Available Boards = "boards board1 board2 board3"

\* Draw = "draw boardName command param1 param2 param3"

\* Example: "draw boardName drawLineSegment x1 y1 x2 y2 color width"

\* Check and add User = "checkAndAddUser username boardName boolean"

\*

\*

\*

\* @param input message from client

\* @return message to client

\* @throws IOException

\*/

private String handleRequest(String input) throws IOException, IllegalArgumentException {

System.out.println(input);

String nameReg = "[a-zA-Z0-9\\.\\:\\[\\]\\,\\=\\\\]+";

String regex = "(boards)|(newBoard "+nameReg+")|"

+ "(switch "+nameReg+" "+nameReg+" "+nameReg+")|"

+ "(exit "+nameReg+")|(users "+nameReg+")|"

+ "(checkAndAddUser "+nameReg+" "+nameReg+")|"

+ "(draw "+nameReg+"( "+nameReg+")+)";

if ( ! input.matches(regex)) {

// invalid input

System.out.println("Invalid input: "+input);

return null;

}

String[] tokens = input.split(" ");

// Get Boards

if (tokens[0].equals("boards")) {

return boards(tokens);

}

//New Board

else if (tokens[0].equals("newBoard")) {

return newBoard(tokens);

}

// Switch Board

else if (tokens[0].equals("switch")) {

return switchBoard(tokens);

}

// Exit

else if (tokens[0].equals("exit")) {

return exit(tokens);

}

// Draw Command

else if (tokens[0].equals("draw")) {

return draw(tokens);

}

// Check and add User

else if (tokens[0].equals("checkAndAddUser")) {

return checkAndAddUser(tokens);

}

// Get Users

else if (tokens[0].equals("users")) {

return users(tokens);

}

// Should never get here-- should return in each of the valid cases above.

throw new UnsupportedOperationException();

}

/\*\*

\* Boards response

\* @param tokens

\* @return

\*/

public String boards(String[] tokens) {

return "boards" + server.getBoards();

}

/\*\*

\* New board response

\* @param tokens

\* @return

\*/

public String newBoard(String[] tokens) {

String boardName = tokens[1];

return "newBoard " + boardName + " " + String.valueOf(server.newBoard(boardName));

}

/\*\*

\* Switch board response

\* @param tokens

\* @return

\*/

public String switchBoard(String[] tokens) {

String userName = tokens[1];

String oldBoardName = tokens[2];

String newBoardName = tokens[3];

String newLine = System.getProperty("line.separator");

server.switchBoard(userName, oldBoardName, newBoardName);

List<Command> commands = server.getCommands(newBoardName);

String str = "switch " + userName + " " + oldBoardName + " " + newBoardName;

for (Command command: commands) {

str += newLine+command.toString();

}

return str;

}

/\*\*

\* Exit board response

\* @param tokens

\* @return

\*/

public String exit(String[] tokens) {

String username = tokens[1];

server.exit(username);

return "exit " + username;

}

/\*\*

\* draw response

\* @param tokens

\* @return

\*/

public String draw(String[] tokens) {

String boardName = tokens[1];

Command command = new Command(tokens);

server.updateBoard(boardName, command);

server.sendCommandToClients(command, socket);

return command.toString();

}

/\*\*

\* checkAndAddUser response

\* @param tokens

\* @return

\*/

public String checkAndAddUser(String[] tokens) {

String boardName = tokens[2];

String username = tokens[1];

return "checkAndAddUser " + username + " " + boardName + " " + String.valueOf(server.checkUser(username, boardName));

}

/\*\*

\* Get Users response

\* @param tokens

\* @return

\*/

public String users(String[] tokens) {

String boardName = tokens[1];

return "users "+boardName+" "+server.getUsers(boardName);

}

}