**Course**: Principals of Software Development – ENSF 409

**Lab 6**

**Instructor**: M. Moshirpour

**Student Name**: Mitchell Sawatzky

**Date Submitted**: March 7, 2016

Exercise B

BlockingPlayer.java

import java.io.BufferedReader;

import java.io.PrintWriter;

/\*\*

\* Provides a tic tac toe robot Player that attempts to block every move it's opponent makes.

\* @version 1.0

\* @author Mitchell Sawatzky and Connor Newman

\* @since Mar 2016

\*/

public class BlockingPlayer extends RandomPlayer {

/\*\*

\* Constructs a BlockingPlayer object with the specified name, mark, and board.

\* @param name the Player's name

\* @param mark the Player's mark

\* @param board the Player's board

\* @param in the incoming socket

\* @param out the outgoing socket

\*/

public BlockingPlayer(String name, char mark, Board board, BufferedReader in, PrintWriter out) {

super (name, mark, board, in, out);

}

/\*\*

\* Detects whether or not the opponent is about to win, and blocks it if necesarry, otherwise it makes a random move.

\*/

protected void makeMove() {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if ((board.getMark(i, j) == SPACE\_CHAR) && testForBlocking(i, j)) {

board.addMark(i, j, mark);

return;

}

}

}

super.makeMove();

}

/\*\*

\* Tests wether or not the specified board space would win the game for the opponent if they played there on the next turn.

\* @param row the row of the board to test

\* @param col the column of the board to test

\* @return true if the space needs to be blocked, false otherwise

\*/

protected boolean testForBlocking(int row, int col) {

char oM = opponent.mark();

boolean res = true;

// row

for (int i = 0; i < 3; i++) {

if ((i != col) && (board.getMark(row, i) != oM)) {

res = false;

break;

}

}

if (res)

return true;

// col

res = true;

for (int i = 0; i < 3; i++) {

if ((i != row) && (board.getMark(i, col) != oM)) {

res = false;

break;

}

}

if (res)

return true;

// can't be diagonal

if ((row + col) % 2 != 0)

return false;

// diagonal

switch (row) {

case 0:

if (col != 0 && board.getMark(2, 0) == oM && board.getMark(1, 1) == oM)

return true;

else if (board.getMark(2, 2) == oM && board.getMark(1, 1) == oM)

return true;

break;

case 1:

if ((board.getMark(0, 0) == oM && board.getMark(2, 2) == oM) ||

(board.getMark(0, 2) == oM && board.getMark(2, 0) == oM))

return true;

break;

case 2:

if (col != 0 && board.getMark(0, 0) == oM && board.getMark(1, 1) == oM)

return true;

else if (board.getMark(0, 2) == oM && board.getMark(1, 1) == oM)

return true;

break;

}

return false;

}

}

Board.java

import java.io.PrintWriter;

// Board.java

// ENSF 409 - LAB 3 - Ex. C

// This file was originally written for ENGG 335 in fall 2001, and was

// adapted for ENSF 409 in 2014

//

/\*\*

\* Provides a tic-tac-toe board and logic to fill, empty, and test if a player has won.

\* @author Originally written by Mahmood Moussavi, modified by Mitchell Sawatzky and Connor Newman

\* @version 1.0

\* @since Originally written in fall 2001, adapted in 2014, modified in 2016

\*/

public class Board implements Constants {

/\*\*

\* Two-Dimensional char array to hold the values of each slot on the board

\*/

private char theBoard[][];

/\*\*

\* The total number of slots filled in on the board.

\*/

private int markCount;

/\*\*

\* Constructs a Board object without any spaces filled in.

\*/

public Board() {

markCount = 0;

theBoard = new char[3][];

for (int i = 0; i < 3; i++) {

theBoard[i] = new char[3];

for (int j = 0; j < 3; j++)

theBoard[i][j] = SPACE\_CHAR;

}

}

/\*\*

\* Returns the value of a board slot at a given row and column.

\* @param row the row to retrieve the board slot from

\* @param col the column to retrieve the board slot from

\* @return the Character value of the board slot

\*/

public char getMark(int row, int col) {

return theBoard[row][col];

}

/\*\*

\* Returns whether or not the board has values in all 9 slots.

\* @return True if all 9 slots are full, False otherwise

\*/

public boolean isFull() {

return markCount == 9;

}

/\*\*

\* Checks whether or not the letter X has won on the current board.

\* @return 0 if X has not won, 1 otherwise

\*/

public int xWins() {

return checkWinner(LETTER\_X);

}

/\*\*

\* Checks whether or not the letter O has won on the current board.

\* @return 0 if O has not won, 1 otherwise

\*/

public int oWins() {

return checkWinner(LETTER\_O);

}

/\*\*

\* Prints the board to stdout.

\* @param out the stream to output the board to

\*/

public void display(PrintWriter out) {

displayColumnHeaders(out);

addHyphens(out);

for (int row = 0; row < 3; row++) {

addSpaces(out);

out.print("P row " + row + ' ');

for (int col = 0; col < 3; col++)

out.print("| " + getMark(row, col) + " ");

out.println("|");

addSpaces(out);

addHyphens(out);

}

}

/\*\*

\* Sets the value of the board slot at a given row and column.

\* @param row the row to set the slot value

\* @param col the column to set the slot value

\* @param mark the Character to set the slot to

\*/

public void addMark(int row, int col, char mark) {

theBoard[row][col] = mark;

markCount++;

}

/\*\*

\* Resets every value on the board to SPACE\_CHAR.

\*/

public void clear() {

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

theBoard[i][j] = SPACE\_CHAR;

markCount = 0;

}

/\*\*

\* Uses tic-tac-toe logic to determine if a specific player has won.

\* @param mark the player to check, either LETTER\_X or LETTER\_O

\* @return 0 if the player has lost, 1 otherwise

\*/

int checkWinner(char mark) {

int row, col;

int result = 0;

for (row = 0; result == 0 && row < 3; row++) {

int row\_result = 1;

for (col = 0; row\_result == 1 && col < 3; col++)

if (theBoard[row][col] != mark)

row\_result = 0;

if (row\_result != 0)

result = 1;

}

for (col = 0; result == 0 && col < 3; col++) {

int col\_result = 1;

for (row = 0; col\_result != 0 && row < 3; row++)

if (theBoard[row][col] != mark)

col\_result = 0;

if (col\_result != 0)

result = 1;

}

if (result == 0) {

int diag1Result = 1;

for (row = 0; diag1Result != 0 && row < 3; row++)

if (theBoard[row][row] != mark)

diag1Result = 0;

if (diag1Result != 0)

result = 1;

}

if (result == 0) {

int diag2Result = 1;

for (row = 0; diag2Result != 0 && row < 3; row++)

if (theBoard[row][3 - 1 - row] != mark)

diag2Result = 0;

if (diag2Result != 0)

result = 1;

}

return result;

}

/\*\*

\* Print the board's column headers to stdout.

\* @param out the stream to output the board to

\*/

void displayColumnHeaders(PrintWriter out) {

out.print("P ");

for (int j = 0; j < 3; j++)

out.print("|col " + j);

out.println();

}

/\*\*

\* Adds a line to separate the board's rows.

\* @param out the stream to output the board to

\*/

void addHyphens(PrintWriter out) {

out.print("P ");

for (int j = 0; j < 3; j++)

out.print("+-----");

out.println("+");

}

/\*\*

\* Adds spacing inside the board to correctly place the values of the slots.

\* @param out the stream to output the board to

\*/

void addSpaces(PrintWriter out) {

out.print("P ");

for (int j = 0; j < 3; j++)

out.print("| ");

out.println("|");

}

}

Constants.java

//Constants.java

/\*\*

\* Provides constants to the rest of the Package.

\* @version 1.0

\* @author Originally written by Mahmood Moussavi, modified by Mitchell Sawatzky and Connor Newman

\* @since Originally written in fall 2001, adapted in 2014, modified in 2016

\*/

public interface Constants {

/\*\*

\* The character to use when the board slot is empty.

\*/

static final char SPACE\_CHAR = ' ';

/\*\*

\* The character to use when Player O has entered into the board.

\*/

static final char LETTER\_O = 'O';

/\*\*

\* The character to use when Player X has entered into the board.

\*/

static final char LETTER\_X = 'X';

}

Game.java

//Game.java

import java.io.\*;

/\*\*

\* @author Started by: M. Moussavi

\* Completed by: Mitchell Sawatzky and Connor Newman

\* Asks the user to select a player type, creates the player, creates the board,

\* assigns a referee to the game, then initiates the game.

\*/

public class Game implements Constants {

/\*\*

\* The board

\*/

private Board theBoard;

/\*\*

\* The referee

\*/

private Referee theRef;

/\*\*

\* creates a board for the game

\*/

public Game( ) {

theBoard = new Board();

}

/\*\*

\* calls the referee method runTheGame

\* @param r refers to the appointed referee for the game

\* @throws IOException

\*/

public void appointReferee(Referee r) throws IOException {

theRef = r;

theRef.runTheGame();

}

/\*\*

\* Creates the specified type of player indicated by the user.

\*

\* @param name player's name

\* @param mark player's mark (X or O)

\* @param board refers to the game board

\* @param sin refers to an input stream

\* @param sout refers to an output stream

\* @return a newly created player

\* @throws IOException

\*/

static public Player create\_player(String name, char mark, Board board,

BufferedReader sin, PrintWriter sout) throws IOException {

// Get the player type.

final int NUMBER\_OF\_TYPES = 4;

sout.println("P \nP What type of player is " + name + "?\nP ");

sout.println("P 1: human\nP " + " 2: Random Player\nP "

+ " 3: Blocking Player\nP " + " 4: Smart Player\nP ");

sout.println("I Please enter a number in the range 1-" + NUMBER\_OF\_TYPES + ": ");

int player\_type = 0;

String input;

input = sin.readLine();

if (input == null || input.length() == 0) {

player\_type = -1;

} else {

player\_type = Integer.parseInt(input);

}

while (player\_type < 1 || player\_type > NUMBER\_OF\_TYPES) {

sout.println("P Please try again.\nP ");

sout.println("I Enter a number in the range 1-" +NUMBER\_OF\_TYPES + ": ");

input = sin.readLine();

if (input == null || input.length() == 0) {

player\_type = -1;

} else {

player\_type = Integer.parseInt(input);

}

}

// Create a specific type of Player

Player result = null;

switch(player\_type) {

case 1:

result = new HumanPlayer(name, mark, board, sin, sout);

break;

case 2:

result = new RandomPlayer(name, mark, board, sin, sout);

break;

case 3:

result = new BlockingPlayer(name, mark, board, sin, sout);

break;

case 4:

result = new SmartPlayer(name, mark, board, sin, sout);

break;

default:

System.out.print ( "\nDefault case in switch should not be reached.\n"

+ " Program terminated.\n");

System.exit(0);

}

return result;

}

/\*\*

\* Starts a new game

\* @param p1sin the input stream for player 1

\* @param p1sout the output stream for player 1

\* @param p2sin the input stream for player 2

\* @param p2sout the output stream for player 2

\* @throws IOException

\*/

public void start (BufferedReader p1sin, PrintWriter p1sout, BufferedReader p2sin, PrintWriter p2sout) throws IOException {

Player xPlayer, oPlayer;

p1sout.println("I Please enter the name of the \'X\' player.");

String name = p1sin.readLine();

while (name == null && name.length() != 0) {

p1sout.println("I Please try again: ");

name = p1sin.readLine();

}

xPlayer = Game.create\_player(name, LETTER\_X, theBoard, p1sin, p1sout);

p2sout.println("I Please enter the name of the \'O\' player.");

name = p2sin.readLine();

while (name == null && name.length() != 0) {

p2sout.println("I Please try again: ");

name = p2sin.readLine();

}

oPlayer = Game.create\_player(name, LETTER\_O, theBoard, p2sin, p2sout);

appointReferee(new Referee(theBoard, xPlayer, oPlayer));

}

}

HumanPlayer.java

import java.io.BufferedReader;

import java.io.PrintWriter;

/\*\*

\* Provides methods to gather input from stdin in order to play a game of tic tac toe.

\* @author Mitchell Sawatzky and Connor Newman

\* @version 1.0

\* @since Mar 2016

\*/

public class HumanPlayer extends Player {

/\*\*

\* Constructs a HumanPlayer object with the specified name, mark, and board.

\* @param name the Player's name

\* @param mark the Player's mark

\* @param board the Board to play the game on

\* @param in the input stream

\* @param out the output stream

\*/

public HumanPlayer(String name, char mark, Board board, BufferedReader in, PrintWriter out) {

super(name, mark, board, in, out);

}

/\*\*

\* Starts a game of tic tac toe with this player as player X.

\* @throws IOException

\*/

public void play() throws java.io.IOException {

String winner;

Player p = this;

while (true) {

if (board.isFull()) {

winner = "Nobody";

break;

} else if (board.xWins() == 1) {

winner = name;

break;

} else if (board.oWins() == 1) {

winner = opponent.name();

break;

}

board.display(p.sout);

p.makeMove();

p = p.opponent;

}

sout.printf("P \nP THE GAME IS OVER: %s is the winner!\n", winner);

opponent.sout.printf("P \nP THE GAME IS OVER: %s is the winner!\n", winner);

sout.println("Q");

opponent.sout.println("Q");

sout.close();

sin.close();

opponent.sout.close();

opponent.sin.close();

System.exit(0);

}

/\*\*

\* Prompts the user via stdout to make a move on the tic tac toe Baord.

\* @throws IOException

\*/

public void makeMove() throws java.io.IOException {

int row, col;

Player p = this;

while (true) {

while (true) {

String input = "";

boolean success = true;

do {

sout.printf("I %s, what row should your next %c be placed in?\n", p.name, p.mark);

input = sin.readLine();

try {

Integer.parseInt(input);

success = false;

} catch (NumberFormatException e) {

}

} while (success);

row = Integer.parseInt(input);

if (row < 0 || row > 2)

sout.printf("P \nP Invalid row: %d, please try again.\n", row);

else

break;

}

while (true) {

String input = "";

boolean success = true;

do {

sout.printf("I %s, what column should your next %c be placed in?\n", p.name, p.mark);

input = sin.readLine();

try {

Integer.parseInt(input);

success = false;

} catch (NumberFormatException e) {

}

} while (success);

col = Integer.parseInt(input);

if (col < 0 || col > 2)

sout.printf("P \nP Invalid row: %d, please try again.\n", col);

else

break;

}

if (board.getMark(row, col) == SPACE\_CHAR) {

board.addMark(row, col, mark);

break;

} else {

sout.printf("P \nP The coordinate (%d, %d) has already been used.\n", row, col);

}

}

}

}

Player.java

//Player.java

import java.io.BufferedReader;

import java.io.PrintWriter;

/\*\*

\* Provides a container to hold a Player's name and preferred mark (X or O), as well as logic prototypes.

\* @author Mitchell Sawatzky and Connor Newman

\* @version 1.0

\* @since Mar 5, 2016

\*/

abstract class Player implements Constants {

/\*\*

\* The name of the player.

\*/

protected String name;

/\*\*

\* The player's mark, either 'X' or 'O'.

\*/

protected char mark;

/\*\*

\* The player's opponent.

\*/

protected Player opponent;

/\*\*

\* The Board to play the game on.

\*/

protected Board board;

/\*\*

\* The input stream

\*/

protected BufferedReader sin;

/\*\*

\* The output stream

\*/

protected PrintWriter sout;

/\*\*

\* Constructs a Player Object with a given name, mark, and Board.

\* @param name the Player's name

\* @param mark the Player's mark, either 'X' or 'O'

\* @param b the Board to play the game on

\* @param in the input stream

\* @param out the output stream

\*/

public Player(String name, char mark, Board b, BufferedReader in, PrintWriter out) {

this.name = name;

this.mark = mark;

this.board = b;

sin = in;

sout = out;

}

/\*\*

\* Getter function for the Player's name.

\* @return the String name of the Player

\*/

protected String name() {

return name;

}

/\*\*

\* Getter function for the Player's mark

\* @return the char mark of the Player

\*/

protected char mark() {

return mark;

}

/\*\*

\* Sets the opponent of a given Player to another Player.

\* @param opp the Player opponent

\*/

protected void setOpponent(Player other) {

this.opponent = other;

}

/\*\*

\* Initiate a game of tic-tac-toe with the opponent player.

\* @throws IOException

\*/

abstract protected void play() throws java.io.IOException;

/\*\*

\* Prompt the user to place their mark on a given board slot retrieved through stdin.

\* @Throws IOException

\*/

abstract protected void makeMove() throws java.io.IOException;

}

RandomGenerator.java

// RandomGenerator.java

import java.util.Random;

/\*\*

\* Provides a method to spawn a random integer.

\* @author M. Moussavi

\*/

class RandomGenerator {

/\*\*

\* creates a random number ranging between lo and hi,

\* @param lo the lower bound of the random integer

\* @param hi the upper bound of the random integer

\* @return the random integer

\*/

int discrete(int lo, int hi)

{

if(lo >= hi){

System.out.println("Error discrete, lo >= hi");

System.exit(0);

}

Random r = new Random();

int d = r.nextInt(hi - lo + 1) + lo;

return d;

}

}

RandomPlayer.java

import java.io.BufferedReader;

import java.io.PrintWriter;

/\*\*

\* Provides a tic tac toe robot that randomly chooses a space on every move.

\* @author Mitchell Sawatzky and Connor Newman

\* @version 1.0

\* @since Mar 2016

\*/

public class RandomPlayer extends Player {

/\*\*

\* Constructs a RandomPlayer object with the specified name, mark, and board.

\* @param name the name of the player

\* @param mark the mark of the player

\* @param board the game board

\* @param in the input stream

\* @param out the output stream

\*/

public RandomPlayer(String name, char mark, Board board, BufferedReader in, PrintWriter out) {

super(name, mark, board, in, out);

}

/\*\*

\* Starts a game of tic tac toe with this player as player X.

\* @throws IOException

\*/

protected void play() throws java.io.IOException {

String winner;

Player p = this;

while (true) {

if (board.isFull()) {

winner = "Nobody";

break;

} else if (board.xWins() == 1) {

winner = p.name;

break;

} else if (board.oWins() == 1) {

winner = p.opponent.name();

break;

}

board.display(p.sout);

p.makeMove();

p = p.opponent;

}

sout.printf("P \nP THE GAME IS OVER: %s is the winner!\n", winner);

opponent.sout.printf("P \nP THE GAME IS OVER: %s is the winner!\n", winner);

sout.println("Q");

opponent.sout.println("Q");

sout.close();

sin.close();

opponent.sout.close();

opponent.sin.close();

System.exit(0);

}

/\*\*

\* Picks a random board slot and makes a move there.

\*/

protected void makeMove() {

RandomGenerator rand = new RandomGenerator();

int row, col;

do {

row = rand.discrete(0, 2);

col = rand.discrete(0, 2);

} while (board.getMark(row, col) != SPACE\_CHAR);

board.addMark(row, col, mark);

}

}

Referee.java

//Referee.java

/\*\*

\* Mediates and controls a game of Tic Tac Toe.

\* Begins the game by printing the board, and then asks Player X to choose

\* @author Mitchell Sawatzky and Connor Newman

\* @version 1.0

\* @since Mar 5, 2016

\*/

public class Referee {

/\*\*

\* Player X of the game.

\*/

private Player x;

/\*\*

\* Player O of the game.

\*/

private Player o;

/\*\*

\* The board to play on.

\*/

private Board b;

/\*\*

\* Construct a Referee object from Players and a Board.

\* @param board the Board for the referee to control

\* @param xPlayer the player with the mark 'X'

\* @param oPlayer the player with the mark 'O'

\*/

public Referee(Board board, Player xPlayer, Player oPlayer) {

this.b = board;

this.x = xPlayer;

this.o = oPlayer;

}

/\*\*

\* Initiate a game with Player X as the starting player.

\* @throws IOException

\*/

public void runTheGame() throws java.io.IOException {

x.setOpponent(o);

o.setOpponent(x);

x.play();

x.sout.println("\033[1mGame ended ...\033[0m");

o.sout.println("\033[1mGame ended ...\033[0m");

}

}

SmartPlayer.java

import java.io.BufferedReader;

import java.io.PrintWriter;

/\*\*

\* Provides a tic tac toe robot that first checks it it can win, and then checks whether or not it can block the opponent from winning.

\* @author Mitchell Sawatzky and Connor Newman

\* @version 1.0

\* @since Mar 2016

\*/

public class SmartPlayer extends BlockingPlayer {

/\*\*

\* Constructs a SmartPlayer object with the specified name, mark, and board.

\* @param name the Player's name

\* @param mark the Player's mark

\* @param board the board to play the game on

\* @param in the input stream

\* @param out the output stream

\*/

public SmartPlayer(String name, char mark, Board board, BufferedReader in, PrintWriter out) {

super(name, mark, board, in, out);

}

/\*\*

\* First checks whether it can win the game, and then falls back to BlockingPlayer's logic to block the opponent.

\*/

protected void makeMove() {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if ((board.getMark(i, j) == SPACE\_CHAR) && testForWinning(i, j)) {

board.addMark(i, j, mark);

return;

}

}

}

super.makeMove();

}

/\*\*

\* Decides whether placing a mark in the specified row and column will win the getName

\* @param row the row to place the mark in

\* @param col the column to place the mark in

\* @return true if placing the mark wins the game, false otherwise

\*/

public boolean testForWinning(int row, int col) {

boolean res = true;

// row

for (int i = 0; i < 3; i++) {

if ((i != col) && (board.getMark(row, i) != mark)) {

res = false;

break;

}

}

if (res)

return true;

// col

res = true;

for (int i = 0; i < 3; i++) {

if ((i != row) && (board.getMark(i, col) != mark)) {

res = false;

break;

}

}

if (res)

return true;

// can't be diagonal

if ((row + col) % 2 != 0)

return false;

// diagonal

switch (row) {

case 0:

if (col != 0 && board.getMark(2, 0) == mark && board.getMark(1, 1) == mark)

return true;

else if (board.getMark(2, 2) == mark && board.getMark(1, 1) == mark)

return true;

break;

case 1:

if ((board.getMark(0, 0) == mark && board.getMark(2, 2) == mark) ||

(board.getMark(0, 2) == mark && board.getMark(2, 0) == mark))

return true;

break;

case 2:

if (col != 0 && board.getMark(0, 0) == mark && board.getMark(1, 1) == mark)

return true;

else if (board.getMark(0, 2) == mark && board.getMark(1, 1) == mark)

return true;

break;

}

return false;

}

}

TTTClient.java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

import java.net.UnknownHostException;

/\*\*

\* Acts an an isolated client designed to interact with the TTTServer

\* @author Mitchell Sawatzky

\* @version 1.0

\* @since Mar 2016

\*/

public class TTTClient {

/\*\*

\* The outgoing stream

\*/

private PrintWriter sOut;

/\*\*

\* The client socket

\*/

private Socket sock;

/\*\*

\* The incoming stream

\*/

private BufferedReader sIn;

/\*\*

\* A BufferedReader of stdin

\*/

private BufferedReader stdIn;

/\*\*

\* Constructs a TTTClient opject

\* @constructor

\*/

public TTTClient () {

try {

sock = new Socket("localhost", 8080);

sIn = new BufferedReader(new InputStreamReader(sock.getInputStream()));

sOut = new PrintWriter(sock.getOutputStream(), true);

stdIn = new BufferedReader(new InputStreamReader(System.in));

sOut.println("R");

} catch (IOException e) {

System.err.println("Unable to connect to localhost:8080");

System.err.println(e.getStackTrace());

}

}

/\*\*

\* Logic for parsing server responses and instructions

\*/

public void parseServer () {

String line = "";

try {

do {

line = sIn.readLine();

// System.out.println("server>> " + line);

if (line != null) {

switch (line.substring(0, 1)) {

case "R": // READY

System.out.println("Connected to the server...");

break;

case "I": // INPUT

System.out.println(line.substring(2, line.length()));

sOut.println(stdIn.readLine());

break;

case "P": // PRINT

System.out.println(line.substring(2, line.length()));

break;

case "S": // Server full

System.out.println(line);

line = "QUIT";

break;

case "Q":

line = "QUIT";

break;

}

} else {

System.out.println("The server disconnected you.");

line = "QUIT";

}

} while (line != "QUIT");

sIn.close();

sOut.close();

stdIn.close();

} catch (IOException e) {

System.err.println(e.getStackTrace());

}

}

/\*\*

\* Program entry point

\* @param argv the command line arguments

\*/

public static void main (String[] argv) {

TTTClient cli = new TTTClient();

cli.parseServer();

}

}

TTTServer.java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

/\*\*

\* Server for a game of Tic Tac Toe

\* @author Mitchell Sawatzky

\* @version 1.0

\* @since Mar, 2016

\*/

public class TTTServer implements Constants {

/\*\*

\* The server socket

\*/

private ServerSocket sSock;

/\*\*

\* A ServerConnection for player 1

\*/

protected ServerConnection p1 = null;

/\*\*

\* A ServerConnection for player 2

\*/

protected ServerConnection p2 = null;

/\*\*

\* Thread wrapper for each connection

\* @author Mitchell Sawatzky

\* @since Mar, 2016

\* @version 1.0

\*/

private class ServerConnection implements Runnable {

/\*\*

\* The output stream

\*/

private PrintWriter sOut;

/\*\*

\* The input stream

\*/

private BufferedReader sIn;

/\*\*

\* The client socket

\*/

private Socket sock;

/\*\*

\* Whether or not this is player 1

\*/

private boolean playerOne;

/\*\*

\* Constructs a ServerConnection ovject with a socket

\* @constructor

\* @param s the client socket

\* @param pOne if this connection is player 1

\* @throws IOException

\*/

private ServerConnection (Socket s, boolean pOne) throws IOException {

sock = s;

sIn = new BufferedReader(new InputStreamReader(s.getInputStream()));

sOut = new PrintWriter(s.getOutputStream(), true);

playerOne = pOne;

}

/\*\*

\* Thread entry point

\*/

public void run () {

try {

String line = "";

do {

try {

line = sIn.readLine();

if (line == null) {

throw new Exception();

}

} catch (Exception e) {

System.out.println((playerOne ? "p1" : "p2") + " disconnected...");

sIn.close();

sOut.close();

if (playerOne) {

p1 = null;

} else {

p2 = null;

}

}

// System.out.println((playerOne ? "p1" : "p2" ) + ">> " + line);

sOut.println("R");

if (playerOne) {

sOut.println("P Waiting for player 2...");

} else {

Game theGame = new Game();

try {

theGame.start(p1.sIn, p1.sOut, sIn, sOut);

} catch (IOException e) {

sIn.close();

sOut.close();

p1.sIn.close();

p1.sOut.close();

p1 = null;

p2 = null;

}

break;

}

} while (line == "R");

} catch (IOException e) {

System.err.println("Error closing connection");

System.err.println(e.getStackTrace());

}

}

}

/\*\*

\* Constructs a TTTServer object

\* @constructor

\*/

public TTTServer () {

try {

sSock = new ServerSocket(8080);

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

} catch (IOException e) {

System.err.println("Could not start server on localhost:8080");

System.err.println(e.getStackTrace());

}

}

/\*\*

\* Program entry point

\* @param argv the command line arguments

\*/

public static void main (String[] argv) {

TTTServer server = new TTTServer();

server.listen();

}

/\*\*

\* Listen for new connections to the server

\*/

public void listen () {

System.out.println("Listening on localhost:8080");

while (true) {

try {

Socket s = sSock.accept();

if (p1 == null) {

p1 = new ServerConnection(s, true);

System.out.println("Player 1 connected");

Thread t = new Thread(p1);

t.start();

} else if (p2 == null) {

p2 = new ServerConnection(s, false);

System.out.println("Player 2 connected");

Thread t = new Thread(p2);

t.start();

} else {

try {

PrintWriter reject = new PrintWriter(s.getOutputStream(), true);

reject.println("Sorry, this server is full.");

s.close();

reject.close();

System.out.println("Rejected a player");

} catch (IOException e) {

System.err.println("Error rejecting connection");

System.err.println(e.getStackTrace());

}

}

} catch (IOException e) {

System.err.println("Error establishing new client");

System.err.println(e.getStackTrace());

}

}

}

}





