/** This class will control a game of checkers between
  players over the Internet by acting as the server. The class allows for
  multiple clients to login to the server using a username and password
  which are saved into a database file. Upon logging in, the server can
  send the client a list of the available players to begin a game with and
  if the player wishes to play and selects an opponent, the server will
  send a message to the opponent asking him if he accepts the challenge
  and if so the server will begin a new game between the two players. The
  checkers game engine is contained within the client class and the server
  is only responsible for passing moves and other communication between the
  two players. If a player has won the client will send a message to the
  server telling the server of this and then the server will send a message
  to the other player, telling that client he has lost. The communication
  between the server and the client is done by sending a 6 character
  OptCode, which tells the server or client what to do, followed by an
  unlimited length parameter which could be a username, checkersmove,
  chat message, etc. */

public class CheckersServer extends Thread{

private ServerSocket server;
private Socket connect;

/** maximum number of players the server will allow.
  MAX_PLAYERS should be one more than the actual
  max number of players (i.e. if you want 10 players,
  MAX_PLAYERS = 11
  */
final int MAX_PLAYERS = 500;

/** (@link PlayerHandler PlayerHandler) array storing all the new clients
  (players) that connect*/
private PlayerHandler players[];
/** Array of size (@link #MAX_PLAYERS MAX_PLAYERS) which tells whether or not
  a slot in the (@link #players players[]) array contains a valid
  player. If the array contains a value of 1, that slot in players
  is being used, otherwise if the array index contains a value of -1,
  that slot in players array is available for a new player
  */
private int emptySlots[];

/** Default constructor to create a new Checkers server */
public CheckersServer()
{
  // set up ServerSocket
  try
server = new ServerSocket( 2999, MAX_PLAYERS );

// process problems creating ServerSocket

try
{
    e.printStackTrace();
    System.exit( 1 );
}

System.out.println( "Server awaiting connections\n" );

/** method which will wait for new clients to connect and establish a
 input/output connection with these new clients */

public void execute()
{

    CheckersGame checkersGame = new CheckersGame();
    players = new PlayerHandler[MAX_PLAYERS];

    int j,i=0;
    emptySlots = new int[MAX_PLAYERS];
    // look for an empty slot in the players array
    // emptySlots array contains a 1 if slot available or
    // -1 if slot is being used by a valid player
    for(j =0;j<MAX_PLAYERS;j++)
    {
        emptySlots[j] = 1; // value of 1 means slot available
    }
    while(true)
    {
        try
        {
            // if i equals the maximum number of players reset i back to 0
            if(i == MAX_PLAYERS)
            {
                i = 0;
                continue;
            }
            // if a slot in players array is available, create a new player
            // to fill that slot
            if(emptySlots[i] == 1)
            {
                players[i] = new PlayerHandler(checkersGame,'R',listen());
                players[i].start();
                emptySlots[i] = -1; // set slot to used
                i++; // increment i
            }
            // if a slot in players is not available, that is there is a current
            // player taking up that slot, check if that player is gone and if
131 // so reset that slot to available and if not increment counter and
132 // check next slot
133 else if(emptySlots[i] == -1)
134 {
135     if(players[i] == null || players[i].isPlayerGone())
136     {
137         emptySlots[i] = 1;
138     }
139     i++; // increment i
140 }
141
142 // catch any exeptions that occur from accepting new clients
143 catch(Exception ex)
144 {
145     ex.printStackTrace();
146     System.exit(1);
147 }
148
149 } // end method execute
150
151 /** this method will "listen" for new clients trying to connect and
152 accept their connection returning the Socket connection they
153 connected at.
154 *@
155 public Socket listen()
156 {
157     Socket TempSocket = null;
158
159     //start serversocket to listen
160     try
161     {
162         System.out.println("Server is listening for incoming connection...");
163         TempSocket = server.accept();
164     }
165     catch (IOException e)
166     {
167         e.printStackTrace(System.out);
168         System.exit(1); // exit if error occurs
169     }
170     return TempSocket;
171 }
172
173 //++++++++++++++++++ Nested Classes +++++++++++++++++
174
175 /** This class stores the {@link PlayerHandler PlayerHandler}
176 for each of the two clients which are currently playing against
177 each other
178 */
179 private class CheckersGame
180 {
181     private PlayerHandler PHandler1 = null;
182     private PlayerHandler PHandler2 = null;
183     /** Default constructor*/
184     public CheckersGame()
185     {
186 }
 /** This method adds a player to the game. If there is no player one, the player is stored as player one, if there is a player one, the player is stored as player two. */
public void AddPlayer(PlayerHandler NewPlayer) {
  if (PHandler1 == null)
    PHandler1 = NewPlayer;
  else
    PHandler2 = NewPlayer;
}

 /** Sets the opponent for each player*/
public void SetOpponent() {
  PHandler1.SetOpponent(PHandler2);
  PHandler2.SetOpponent(PHandler1);
}

 /** checks to see if the current game is full, that is has two players to play against each other
  @return <code>true</code> if there are two players already set or <code>false</code> if there are 0 or 1 players set
 */
public boolean IsFull() {
  return ((PHandler1 != null) && (PHandler2 != null));
}

 /** Get player 1
  @return returns player 1 in the current game
 */
public PlayerHandler GetPlayer1() {
  return PHandler1;
}

 /** Get player 2
  @return returns player 2 in the current game
 */
public PlayerHandler GetPlayer2() {
  return PHandler2;
}

 /** This class will handle all communication of a client between the client and the server as well as store basic information about the client. Such as, is the client still connected and is the client currently playing a game, is he logged in, etc. */
private class PlayerHandler extends Thread {
  /** Current opponent player is playing against*/
  private PlayerHandler Opponent;
  /** PrintWriter to send output to the clients */
  private PrintWriter output = null;
  /** BufferedReader to accept input from the clients */
  private BufferedReader input = null;
  /** Socket to connect to clients over*/
private Socket socket;
/** {@link RandomAccessPlayerRecord RandomAccessPlayerRecord} to record
    data about the current player to: username, password, wins, losses
*/
private RandomAccessPlayerRecord playerRecord;

/** current game player is playing in*/
private CheckersGame game;
/** which gamepiece is the current player currently playing as?
    'R' for red or 'B' for black */
public char sign;
/** name of current player*/
private String MyName;
/** has the current player already logged in */
public boolean loggedIn = false;
/** has the current player disconnected */
private boolean playerGone = true;
/** is the current player currently in a game*/
public boolean playingGame = false;

CheckersGame checkersGame;

/** Constructor to create a new player*/
public PlayerHandler(CheckersGame gm, char tempsign, Socket clientSocket)
    throws Exception
{
    game = gm;
    sign = tempsign;

    try {
        //client connects and we setup clientsocket to communicate
        //with client
        socket = clientSocket;
        output = new PrintWriter(clientSocket.getOutputStream(), true);
        input = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        // send message to client that connection was successful
        output.println("WELCOM");

    } catch (IOException e)
    {
        System.out.println("Error with player connection occurred");
    }
    playerGone = false;

    } /* return current opponent player is playing against*/
public PlayerHandler GetOpponent()
{
    return Opponent;
}

/** set the opponent player is playing against*/
public void SetOpponent(PlayerHandler NewPlayer)
{
    Opponent = NewPlayer;
    // send message telling client an opponent has arrived
    output.println("OPPARR"+NewPlayer.getName());
}

/** get the output stream for the current player*/
public PrintWriter getOutputStream()
{
    if(output != null)
        return output;
    else
        return null;
}

/** is the player still there, or has he disconnected*/
public boolean isPlayerGone()
{
    return playerGone;
}

public void setPlayerGone(boolean temp)
{
    playerGone = temp;
}

/** return the current game that the player is involved in */
public CheckersGame getCurrentGame()
{
    return checkersGame;
}

/** start the thread to listen for messages from Clients */
public void run()
{
    String Command = null; //command = optcode + param
    String OptCode = null;
    String Param = null;
    String tempUsername = null;
    String tempPassword = null;
    boolean haveUsername = false;
    boolean havePassword = false;
    int fromRow;
    int fromCol;
    int toRow;
    int toCol;

    try {
        // continue looping listening for commands from clients
        while (true)
        {
            Command = input.readLine();
            /* Command is a list full of Optcodes received from Clients
            OptCodes are 6 character commands telling server what to do
            followed by a string Param which could be a username or checkers
             move or message to send to opponent ect.
            The OptCodes received from Clients will be:
            
            CONNCT = client acknowledging connct
            USRNAM = client sending username
            PASWRD = client sending password
            NEWMOV = client sending new move
            REDWIN = red has won the game
            BLKWIN = black has won the game
            FLYRLS = client requesting list of connected and logged in players
            MESSGE = server sending message to client
            OPFFREQ = client sending request for a game against certain
            */
opponent

ACCEPT = the opponent has accepted the request to play a game
DECLIN = the opponent has declined the request to play a game
GETSTA = the client has requested that the server sends his
player statistics
RESIGN = the opponent has resigned in the current game
RESACK = the other player has acknowledged the resignation
CHATSE = the client has sent a chat message to the opponent
NUSROK = the client acknowledges that he wants to create new user
LOSTOP = client has disconnected

The following OptCode may be sent by the server throughout the
code:
WELCOM = server has made successful connection with client
OPPARR = opponent for new game has arrived this also signals
the applet to start a new game
INCPSW = login name found, but password for that name incorrect
USRNOK = username and password OK, successful login
YRSIGN = server has set the client to either red or black
MOVETO = server command to move chip to new location
MESSGE = message coming from server
SENDLS = server sending player list
GAFULL = server is full (can't accept more clients)
GAMEQ = another client has challenged you to a game
NOGAME = opponent has declined your challenge to play game
RETSTA = server sending number of wins and losses
OPPRES = opponent resigned game, you win
CHATRE = chat message sent from opponent during game
NEWUSR = your username does not exist and the server is prompting
you to create a new user
OPFGON = opponent has disconnected from server
ALRLOG = server telling you, you are already logged on

if (Command != null)
{
    OptCode = Command.substring(0, 6);
    Param = Command.substring(6).trim();
}
// check the optcode to see what to do
/* This OptCode just signals that the WELCOM transmission was
   successfully received by the client and the client has
   acknowledged that transmission.
   Parameter = null
*/
if(OptCode.equals("CONNCT"))
{
    //System.out.println("transmission successful"); // DEBUG
}
/* This OptCode signals that the client will be sending the username
entered by the user.
   Parameter = username
*/
else if (OptCode.equals("USRNAM"))
{
    tempUsername = new String(Param); // set the temp username for
    // checking later
    haveUsername = true; // we now have a username from client
}
/* This OptCode signals that the client will be sending the password
entered by the user.
Parameter = password

else if(OptCode.equals("PASWRD"))
{
    tempPassword = new String(Param); // set the temp password for
    // checking later
    havePassword = true; // we now have a password from client
}
/* This OptCode signals that the client has made a new checkers
move, all error checking is done by the client, so
the move is known to be valid and will be passed on
to the opponent player
Parameter = checkers move (four integers with
coordinates for move from and
move to i.e. [1,1] to [2,2])
*/
else if(OptCode.equals("NEWMOV"))
{
    String temp;
    // parse the four integers sent as the parameter
    temp = new String(Param.substring(0,1));
    fromRow = Integer.parseInt(temp);
    temp = new String(Param.substring(2,3));
    fromCol = Integer.parseInt(temp);
    temp = new String(Param.substring(4,5));
    toRow = Integer.parseInt(temp);
    temp = new String(Param.substring(6,7));
    toCol = Integer.parseInt(temp);
    // send the move to the opponent
    GetOpponent().sendMove(fromRow,fromCol,toRow,toCol);
}
/* This Optcode signals that the red player has won the game which
is determined in the CheckersClient code
Parameter = null
*/
else if(OptCode.equals("REDWIN"))
{
    // if the current player is red
    if(sign == 'R')
    {
        // the current player has won, so add 1 to his win total
        playerRecord.setNumberWins(playerRecord.getNumberWins()+1);
        // save that new record
        playerRecord.writeRecord();
        // send him a message, stating that he won
        output.println("MESSAGE"+"Congrats, You Won!");
    }
    // if the current player is black
    else if(sign == 'B')
    {
        // the current player has lost, so add 1 to his loss total
        playerRecord.setNumberLosses( // continued
            playerRecord.getNumberLosses()+1);
        // save that new record
        playerRecord.writeRecord();
        // send him a message, stating that he lost
        output.println("MESSAGE"+"I'm sorry, but you Lost");
    }
    playingGame = false; // player no longer playing a game
/* This Optcode signals that the black player has won the game which
is determined in the CheckersClient code
Parameter = null

*/
else if(OptCode.equals("BLKWIN"))
{
    // if the current player is black
    if(sign == 'B')
    {
        // the current player has won, so add 1 to his win total
        playerRecord.setNumberWins(playerRecord.getNumberWins()+1);
        // save that record
        playerRecord.writeRecord();
        // send him a message telling him he won
        output.println("MESSAGE" +"Congrats, You Won!");
    }
    // if the current player is red
    else if(sign == 'R')
    {
        // the current player has lost, so add 1 to his loss total
        playerRecord.setNumberLosses(playerRecord.getNumberLosses()+1);
        // save that record
        playerRecord.writeRecord();
        // send him a message telling him he lost
        output.println("MESSAGE" +"I'm sorry, but you Lost");
    }
    playingGame = false; // player no longer playing a game
}

/* This OptCode signals that the client would like a list of all the
players available to play a game against or to send a chat
message to. For the list of all players to play against, only
those not playing a game will be listed, for the list of all
players to send a message to, all players logged in will
be listed
Parameter = null
*/
else if(OptCode.equals("PLYRLS"))
{
    StringBuffer listPlayers = new StringBuffer();
    // this loop goes through the list of players and adds the
    // username of all players who are currently
    // logged in and who are not playing a game and
    // sends this list to the client. The names in the list
    // a seperated by a " ",
    for(int i = 0; i < MAX_PLAYERS; i++)
    {
        // if the player is not empty
        if(players[i] != null)
        {
            // if the request is for a list of available players
            // to challenge then only the players who are not
            // playing a game should be listed, but if the
            // request is to send player a message than
            // all players logged in should be listed
            if(players[i].loggedIn && !players[i].playingGame)
            {
                listPlayers.append(" " + players[i].getName());
            }
            else if(Param.equals("Chat") && players[i].loggedIn)
            {
                listPlayers.append(" " + players[i].getName());
            }
        }
    }
}
// make a string out of the list of players
String sendParam = new String(listPlayers);
if(Param.equals("Challenge"))
{
    output.println("SENDLS"+sendParam); // send to client
}
else if(Param.equals("Chat"))
{
    output.println("LISCHA"+sendParam);
}

/* This OptCode signals that the client has chosen a player to play from the list of available players.
   Parameter = username of player to challenge */
else if(OptCode.equals("OPPREQ"))
{
    checkersGame = new CheckersGame(); // create a new checkersgame
    checkersGame.AddPlayer(this); // add the current (challenging player) to the game
    // send challenge to Opponent client along with name of challenger
    findPlayerOutputStream(Param).println("GAMERQ"+getName());

    /* this Optcode signals that the client has accepted a challenge from another client and therefore the server should start a game between this client and the challenging client.
       Parameter = username of player who made the original challenge */
    else if(OptCode.equals("ACCEPT"))
    {
        // get the output stream of the challenging player
        // his sign will be red because the player making the challenge is always red
        findPlayerOutputStream(Param).println("YRSIGN"+"R");
        // set the sign of the challenging player to 'R'
        findPlayer(Param).sign = 'R';
        // tell the challenged client his sign is black 'B'
        output.println("YRSIGN"+"B");
        // set his sign to 'B'
        sign = 'B';
        // add the current (challenged player) to the game already created as the second player
        findPlayer(Param).getCurrentGame().AddPlayer(this);

        // check that the game is full (2 players present) (this should always be the case)
        if(findPlayer(Param).getCurrentGame().IsFull())
        {
            // set the opponents in that game
            findPlayer(Param).getCurrentGame().SetOpponent();
        }

        playingGame = true; // the challenged player is now playing a game
        findPlayer(Param).playingGame = true; // the challenging player is now playing

    } /* This optcode signals that the client has declined a challenge by another player.
       Parameter = username of challenging player */
else if(OptCode.equals("DECLIN"))
{
    // send message to challenging player telling him that his
    // challenge has been declined
    findPlayerOutputStream(Param).println("NOGAME"+getName());
}
/* This OptCode signals that the client requests the server to send
him his number of wins and losses
Parameter = null */
else if(OptCode.equals("GETSTA"))
{
    // send client number of wins and number of losses
    output.println("RETSTA"+playerRecord.getNumberWins()+
           " "+playerRecord.getNumberLosses());
}
/* This OptCode signals that the client has resigned in the current
    game. The server should then signal to the opponent client that
    he has won the game.
Parameter = username of opponent */
else if(OptCode.equals("RESIGN"))
{
    playingGame = false; // game over, no longer playing
    // add to clients number of losses
    playerRecord.setNumberLosses(playerRecord.getNumberLosses()+1);
    // save player record
    playerRecord.writeRecord();
    // tell opponent, client has resigned and he wins
    findPlayerOutputStream(Param).println("OPPRES");
    /* This OptCode signals that the client has acknowledged that his
        opponent has resigned.
        Parameter = null */
else if(OptCode.equals("RESACK"))
{
    playingGame = false;
    playerRecord.setNumberWins(playerRecord.getNumberWins()+1);
    playerRecord.writeRecord();
}
/* This OptCode signals that the client is sending his opponent
    a chat message.
Parameter = username of person to send message to + the chat
    message */
else if(OptCode.equals("CHATSE"))
{
    // split the username and message up
    String[] temp = Param.split(" ",2);
    // send the message along with name of sender
    findPlayerOutputStream(temp[0]).println("CHATRE"+getName()+
           " "+temp[1]);
    /* This OptCode signals that the user would like to create a new
    username and password
    Parameter = null */
else if(OptCode.equals("NUSROK"))
{
    // create a new player record
    playerRecord.addNewRecord(playerRecord);
    loggedIn = true; // user is logged in
    // set the username
    setName(playerRecord.getUsername());
// acknowledge new user created and logged in
output.println("USRNOK"+playerRecord.getUsername());

/* This OptCode signals that the client has disconnected
Parameter = null */
else if(OptCode.equals("LOSTOP"))
{
    playingGame = false;
}
/* This OptCode signals that the client has logged out
Parameter = null */
else if(OptCode.equals("LOGOUT"))
{
    socket.close(); // close the communications socket
    setPlayerGone(true);
    loggedIn = false;
}

// has server received both the username and password
// then they should be proccessed
if(haveUsername && havePassword)
{
    // create new player record object
    playerRecord = new RandomAccessPlayerRecord();
    // check if player already exists
    if(playerRecord.findRecord(tempUsername,tempPassword))
    {
        // if the user entered password is not equal to the saved
        // password
        if(!playerRecord.getPassword().equals(tempPassword))
        {
            // tell client the password is incorrect
            output.println("INCPSW"+tempUsername);
        }
        // check list of players to see if the player is already
        // logged in
        else
        {
            boolean alreadyLoggedIn = false;
            for(int k = 0;k < MAX_PLAYERS;k++)
            {
                if(players[k] != null)
                {
                    if(!players[k].isPlayerGone())
                    {
                        if(players[k].getName().equals(tempUsername))
                        {
                            alreadyLoggedIn = true;
                        }
                    }
                }
            }
            // if the player isn't logged in, then tell the client
            // that his log in was successful
            if(!alreadyLoggedIn)
            {
                // send client successful login
                output.println("USRNOK"+playerRecord.getUsername());
                loggedIn = true; // player logged in
                setName(tempUsername); // set player's name
            }
        }
    }
    // if the player is already logged in (probably shouldn't happen) else {
    output.println("ALRLOG"); // send client message telling him logged in
    alreadyLoggedIn = false; // reset boolean
    }

    // if player record doesn't exist create a new player record
    // with username and password and 0 wins/losses
    else {
        playerRecord.setUsername(tempUsername);
        playerRecord.setPassword(tempPassword);
        playerRecord.setNumberWins(0);
        playerRecord.setNumberLosses(0);
        // send player message asking if he wants to create a new user
        output.println("NEWUSR"+playerRecord.getUsername()+" "+tempPassword);
    }

    haveUsername = false; // reset booleans
    havePassword = false;

}

// catch the exceptions
// an exception will occur when a client is disconnected catch (IOException e) {
    System.out.println("Player has disconnected");
    setPlayerGone(true); // player is now gone
    loggedIn = false; // player is no longer logged in
    // if the disconnected player was playing a game,
    // the opponent will be sent a message telling him
    // his opponent disconnected
    if(playingGame)
        {
            findPlayerOutputStream(GetOpponent().getName()).println("OPPGON");
        }

    }

    /** return the output stream for the player who has the passed username
    * @param name username of the player to find the output stream for
    */
    public PrintWriter findPlayerOutputStream(String name)
    {
        int i;
        // search through list of players for a player with the passed username
        for(i =0;i < MAX_PLAYERS;i++)
        {
            // if the player in the list isn't null or is gone
            // check if the username is the same as the passed name
            if(players[i] != null && !players[i].isPlayerGone())
                {
                    if(name.equals(players[i].getName()))
                        {
                            break;
                        }
846   }  
847   }  
848   // return the output stream if it exists  
849   if(players[i].getOutputStream() != null)  
850   {  
851     return players[i].getOutputStream();  
852   }  
853   else  
854   {  
855     return null;  
856   }  
857  }  
858  }  
859  /** find the PlayerHandler for the player  
860   with the passed username  
861   @param name username of the player to search for  
862   @return type PlayerHandler of the player with the username name  
863   */  
864  public PlayerHandler findPlayer(String name)  
865  {  
866    int i;  
867    for(i =0;i < MAX_PLAYERS;i++)  
868    {  
869      if(players[i] != null && !players[i].isPlayerGone())  
870      {  
871        if(name.equals(players[i].getName()))  
872        {  
873          break;  
874        }  
875      }  
876    }  
877    if(players[i].getOutputStream() != null)  
878    {  
879      return players[i];  
880    }  
881    else  
882    {  
883      return null;  
884    }  
885  }  
886  /** This method formats a checkers move to send it to the opponent client.  
887   4 integers are passed to this method in the row, column where the move  
888   starts to the row, column to where the move should go to  
889   */  
890  public void sendMove(int fromRow,int fromCol,int toRow,int toCol)  
891  {  
892    // send move to opponent  
893    output.println("MOVETO"+fromRow+" "+fromCol+" "+toRow+" "+toCol);  
894  }  
895  }// end class PlayerHandler  
896  
897  /** execute application*/  
898  public static void main( String args[] )  
899  {  
900    CheckersServer application = new CheckersServer();  
901    application.execute();  
902  }  
903  
904}  // end class CheckersServer