JPWP DnDApp - Czat grupowy - Odpowiedzi

Anna Kłucjasz, Krzysztof Skoś

April 2020

Spis treści

1 Odpowiedzi do zadań

We wszytskich zadania zakładamy, że używane zmienne zostały zadeklarowane poprawnie.

1.1 Protokół komunikacyjny

Zaproponuj własną implementację przedstawionego w prezentacji protokołu komunikacyjnego oraz uzupełnij brakujące linijki oznaczone przez "???".

```
private void clientListeningSocket() {
    try {
        socket = new Socket (serverIP , port);
        out = new PrintWriter(socket.getOutputStream(), true);
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        boolean loop = true;
        while(loop){
            final String incomingLine = in.readLine();
            final String displayLine = organizingProtocol(incomingLine);
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    chatMessages.append("\n" + displayLine);
                }
            });
        }
    } catch (IOException e){
        Log.d("tag", Objects.requireNonNull(e.getMessage()));
    }
}
public void onClick (final View view) {
    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            switch (view.getId()) {
                case R.id.sendBtn:
                    String outgoingChatMessage = editMessage.getText().toString();
                    outgoingChatMessage = "MSG" + separator + selectedLanguageID ...
                    ...+ separator + nick + separator + outgoingChatMessage;
                    out.println(outgoingChatMessage);
                    editMessage.setText("");
                    break;
                case R.id.d20Btn:
                    String outgoingDiceRoll = Integer.toString(mkDiceRoll(20));
                    outgoingDiceRoll = "DIC" + separator + 20 + separator ...
                    ...+ nick + separator + outgoingDiceRoll;
                    out.println(outgoingDiceRoll);
                    break;
                case R.id.d4Btn:
                    outgoingDiceRoll = Integer.toString(mkDiceRoll(4));
                    outgoingDiceRoll = "DIC" + separator + 4 + separator ...
                    ...+ nick + separator + outgoingDiceRoll;
                    out.println(outgoingDiceRoll);
                    break;
            }
        }
    });
    thread.start();
}
```

```
private String organizingProtocol (String incomingLine) {
    String[] tmp = incomingLine.split(separator, 4);
    final String statement = tmp[0];        //MSG lub DIC
        switch (statement){
            case "MSG":
            final String receivedLanguageID = tmp[1];
            final String senderNick = tmp[2];
            String message = tmp[3];
            int counter = 0;
            availableLanguages = getAvailableLanguages();
            for (int i = 0; i < availableLanguages.length; i++){
                if (Integer.parseInt(receivedLanguageID) == i && availableLanguages[i]) {
                    counter = 1;
                }
            }
            if (counter == 0) {
                StringBuilder tmpMessage = new StringBuilder();
                String[] pattern = {"~","!","@","#","$","%","^","&","*","+","=","<",">"};
                Random generator = new Random();
                    for (int i = 0; i < message.length(); i++) {
                    tmpMessage.append(pattern[generator.nextInt(pattern.length)]);
                }
                message = tmpMessage.toString();
            }
            return senderNick +": " + message;
            case "DIC":
            final String diceType = tmp[1];
            final String rollerNick = tmp [2];
            final String roll = tmp[3];
                return rollerNick + " wyrzucil " + roll + " na d" +diceType +"!";
    }
    return "";
}
private int mkDiceRoll(int diceType){
    Random generator = new Random();
    return generator.nextInt(diceType) + 1;
}
```

1.2 Stworzenie prostego klienta

Napisz prostą implementację klienta, który komunikuje się z serwerem w sieci lokalnej, mającym IP 192.168.1.4 i nasłuchującym portem 23745.
Ponadto klient powinien móc wysyłać krótkie wiadomości tekstowe serwerowi i czekać na odpowiedź.
Podpowiedź: możesz wykorzystać klasy Socket, BufferedReader, PrintWriter.

Przykład rozwiązania:

```
try {
    Socket client = new Socket ("192.168.1.4", 23745);
    out = new PrintWriter(client.getOutputStream(), true);
    in = new BufferedReader(new InputStreamReader(client.getInputStream()));
    while(true){
        String line = in.readLine();
        chatMessages.append(line);
    }
} catch (IOException e){
    e.printStackTrace();
}
```

1.3 Stworzenie prostego serwera

Napisz prostą implementację serwera z pierwszego zadania. Załóż, że serwer ma ip 192.168.1.4 i powinien nasłuchiwać na porcie 23745. Jeśli uda się połączyć z klientem, to serwer powinien oczekiwać na wiadomość tekstową i odpisywać klientowi "Wiadomość została przyjęta."

```
try {
    ServerSocket server = new ServerSocket (23745);
    Socket client = server.accept();
    out = new PrintWriter(client.getOutputStream(), true);
    in = new BufferedReader(new InputStreamReader(client.getInputStream()));
    while(true){
        String line = in.readLine();
        out.println("Wiadomosc zostala przyjeta.");
    }
} catch (IOException e){
    e.printStackTrace();
}
```

1.4 Implementacja obsługi wielu klientów

Dostosuj serwer z poprzedniego zadania tak, aby serwer mógł rozmawiać równocześnie z wieloma klientami. Przykładowe rozwiązanie:

```
public void servListenSocket () {
    try {
        server = new ServerSocket(port);
    } catch (IOException e) {
        e.printStackTrace();
    }
    while (true) {
        CliWork w;
        try {
            w = new CliWork(server.accept());
            Thread t = new Thread(w);
            t.start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
///////////////////////////////////////
public class CliWork implements Runnable {
    private Socket client;
    private PrintWriter out = null;

    CliWork (Socket client) {
        this.client = client;
    }

    @Override
    public void run() {
        String line;
        BufferedReader in = null;
        PrintWriter out = null;
        try {
            in = new BufferedReader(new InputStreamReader(client.getInputStream()));
            out = new PrintWriter(client.getOutputStream(), true);
        } catch (IOException e) {
            e.printStackTrace();
        }
        while (true) {
            try {
                line = in.readLine();
                out.println("Wiadomosc zostala przyjeta.");
            } catch (IOException e) {
                Log.d("tag", e.getMessage());
            }
        }
    }
}
```

1.5 Czat grupowy

Uzupełnij kod w taki sposób, aby zapewnić komunikację pomiędzy kilkoma klientami połączonymi z serwerem.

```
public class CliWork implements Runnable {
    private Socket client;
    private TextView textArea;
    private PrintWriter out = null;
    private ChatActivity chatActivity;
    private static List<PrintWriter> klienci = new ArrayList<>();

CliWork (Socket client, TextView textArea, ChatActivity chatActivity) {
        this.client = client;
        this.textArea = textArea;
        this.chatActivity = chatActivity;
    }

    public PrintWriter getOut () throws IOException {
        return new PrintWriter(client.getOutputStream(), true);
    }

    @Override
    public void run() {
        String receivedFromClientLine;
        BufferedReader in = null;

        try {
            in = new BufferedReader(new InputStreamReader(client.getInputStream()));
        } catch (IOException e) {
            Log.d("tag", e.getMessage());
        }
        while (true) {
            try {
                assert in != null;
                receivedFromClientLine = in.readLine();
                klienci = ChatActivity.getListOfCliOuts();
                for (int i = 0; i < klienci.size(); i++) {
                    klienci.get(i).println(receivedFromClientLine);
                }
                final String finalReceivedFromClientLine = receivedFromClientLine;
                chatActivity.runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        textArea.append("\n" + finalReceivedFromClientLine);
                    }
                });
            } catch (IOException e) {
                Log.d("tag", e.getMessage());
            }
        }
    }
}
```

```
public class ChatActivity extends AppCompatActivity{

    private String nick;
    private String serverIP;
    private String separator = "@";
    private int port;
    private int selectedLanguageID;
    private boolean[] availableLanguages;
    private EditText editMessage;
    private TextView chatMessages;
    private Socket socket;
    private PrintWriter out;
    private BufferedReader in;
    private ServerSocket server;
    private List<CliWork> listOfClients = new ArrayList<>();
    private static List<PrintWriter> listOfCliOuts = new ArrayList<>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_chat);
        extraInfo();
        addSpinner(mkPersonalLanguageList());
        editMessage = findViewById(R.id.chatETxt);
        chatMessages = findViewById(R.id.replyFromServer);

        Thread thread = new Thread(new Runnable() {
            @Override
            public void run() {
                if (isHost()) {
                    serverListeningSocket();
                } else {
                    clientListeningSocket();
                }
            }
        });
        thread.start();
    }
    //Socket dla servera
    private void serverListeningSocket() {
        try {
            server = new ServerSocket(port);
        } catch (IOException e) {
            Log.d("tag", Objects.requireNonNull(e.getMessage()));
        }
        while (true) {
            CliWork work;
            try {
                work = new CliWork(server.accept(), chatMessages, this);
                listOfClients.add(work);
                listOfCliOuts.add(work.getOut());
                Thread t = new Thread(work);
                t.start();
            } catch (IOException e) {
                Log.d("tag", Objects.requireNonNull(e.getMessage()));
            }
        }
```

```
}
//Socket dla klienta
private void clientListeningSocket() {
    try {
        socket = new Socket (serverIP, port);
        out = new PrintWriter(socket.getOutputStream(), true);
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        boolean loop = true;
        while(loop){
            final String incomingLine = in.readLine();
            final String displayLine = organizingProtocol(incomingLine);
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    chatMessages.append("\n" + displayLine);
                }
            });
        }
    } catch (IOException e){
        Log.d("tag", Objects.requireNonNull(e.getMessage()));
    }
}

public void onClick (final View view) {
    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            switch (view.getId()) {
                case R.id.sendBtn:
                    String outgoingChatMessage = editMessage.getText().toString();
                    outgoingChatMessage = "MSG" + separator + selectedLanguageID ...
                    ...+ separator + nick + separator + outgoingChatMessage;
                    out.println(outgoingChatMessage);
                    editMessage.setText("");
                    break;
                case R.id.d20Btn:
                    String outgoingDiceRoll = Integer.toString(mkDiceRoll(20));
                    outgoingDiceRoll = "DIC" + separator + 20 + separator ...
                    ...+ nick + separator + outgoingDiceRoll;
                    out.println(outgoingDiceRoll);
                    break;
                case R.id.d4Btn:
                    outgoingDiceRoll = Integer.toString(mkDiceRoll(4));
                    outgoingDiceRoll = "DIC" + separator + 4 + separator ...
                    ...+ nick + separator + outgoingDiceRoll;
                    out.println(outgoingDiceRoll);
                    break;
            }
        }
    });
    thread.start();
}

private String organizingProtocol (String incomingLine) {
    String[] tmp = incomingLine.split(separator, 4);
    final String statement = tmp[0];        //MSG lub DIC
    switch (statement){

        case ”MSG”:
            final String receivedLanguageID = tmp[1];
            final String senderNick = tmp[2];
            String message = tmp[3];
            int counter = 0;
            availableLanguages = getAvailableLanguages();
            for (int i = 0; i < availableLanguages.length; i++){
                if (Integer.parseInt(receivedLanguageID) == i && availableLanguages[i]) {
                    counter = 1;
                }
            }
            if (counter == 0) {
                StringBuilder tmpMessage = new StringBuilder();
            String[] pattern = {”~”,”!”,”@”,”#”,”$”,”%”,”^”,”&”,”*”,”+”,”=”,”<”,”>”};
                Random generator = new Random();

                for (int i = 0; i < message.length(); i++) {
                    tmpMessage.append(pattern[generator.nextInt(pattern.length)]);
                }
                message = tmpMessage.toString();
            }
            return senderNick +”: ” + message;

        case ”DIC”:
            final String diceType = tmp[1];
            final String rollerNick = tmp [2];
            final String roll = tmp[3];

            return rollerNick + ” wyrzucil ” + roll + ” na d” +diceType +”!”;
    }
    return ””;
}
private int mkDiceRoll(int diceType){
    Random generator = new Random();
    return generator.nextInt(diceType) + 1;
}


private void addSpinner(final String[] finalLanguageList) {
    Spinner spinner = findViewById(R.id.spinner_jezyki);
    final int[] finalLanguageID = new int[finalLanguageList.length];
    final String[] finalLanguageShow = new String[finalLanguageList.length];

    for (int i = 0; i < finalLanguageList.length; i++){
        String[] tmp = finalLanguageList[i].split(separator, 2);
        finalLanguageID[i] = Integer.parseInt(tmp[0]);
        finalLanguageShow[i] = tmp[1];
    }
    ArrayAdapter<String> adapter = new ArrayAdapter<>(this, ....
    ...android.R.layout.simple_spinner_item, finalLanguageShow);
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    spinner.setAdapter(adapter);
    spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {

        @Override
        public void onItemSelected(AdapterView<?> arg0, View arg1,
                                   int choiceID, long position) {
            selectedLanguageID = finalLanguageID[(int)position];
            Toast.makeText(ChatActivity.this, "Wybrales jszyk: " +...
            ...(finalLanguageShow[choiceID]), Toast.LENGTH_SHORT).show();
        }

        @Override
        public void onNothingSelected(AdapterView<?> arg0) {
            selectedLanguageID = finalLanguageID[0];
        }
    });
}
private String[] mkPersonalLanguageList(){
    String[] allLanguageList = getResources().getStringArray(R.array.languages_array);

    if (getIntent().getBooleanExtra("is_host", false)) {
        return allLanguageList;
    } else {
        availableLanguages = getIntent().getBooleanArrayExtra("avaliableLanguages");
        int counter = 0;
        assert availableLanguages != null;
        for (boolean availableLanguage : availableLanguages) {
            if (availableLanguage) {
                counter++;
            }
        }
        String[] languagePersonalList = new String[counter];
        counter = 0;
        for (int i = 0; i < availableLanguages.length; i++) {
            if (availableLanguages[i]) {
                languagePersonalList[counter] = allLanguageList[i];
                counter++;
            }
        }
        return languagePersonalList;
    }
}
private void extraInfo(){
    nick = getIntent().getStringExtra("nick");
    TextView textView = findViewById(R.id.nickInfoTxt2);
    textView.setText(nick);

    String port_tmp = getIntent().getStringExtra("port");
    textView = findViewById(R.id.portInfoTxt2);
    textView.setText(port_tmp);
    assert port_tmp != null;
    port = Integer.parseInt(port_tmp);

    serverIP = getIntent().getStringExtra("servIP");
    textView = findViewById(R.id.IPinfoTxt2);
    textView.setText(serverIP);
}
private boolean[] getAvailableLanguages(){
    return getIntent().getBooleanArrayExtra("avaliableLanguages");
}
private boolean isHost() {
        return getIntent().getBooleanExtra("is_host", false);
    }
    public static List<PrintWriter> getListOfCliOuts() {
        return  listOfCliOuts;
    }
}
```