Labb 3

Sina Rajaeeian, Malte Bandmann

November 2024

Contents

1 Server4713

⟨*Server4713.java*⟩≡
```
/*****************************************
OBS! No swedish letters in this program.
STEN, SAX and PASE is played.
STEN = ROCK, SAX = SCISSORS, PASE = PAPER
*****************************************/
```
⟨*ImportsServer4713*⟩

⟨*Server4713*⟩

⟨*ClientHandler*⟩

1.1 Imports

Här importeras de klasser som behövs för att skapa en server som lyssnar på port 4713 och hanterar klienter som ansluter till servern.

⟨*ImportsServer4713*⟩≡
```
import java.net.*;
import java.io.*;
import java.util.*;
```

1.2 Server4713

Här skapas en server som lyssnar på port 4713 och hanterar klienter som ansluter till servern. Den skapar en ny tråd för varje klient som ansluter till servern och denna tråd hanterar kommunikationen med klienten. Detta sker helt obereonde av andra klienter som ansluter till servern.

⟨*Server4713*⟩≡
```
public class Server4713 {
    public static void main( String[] args) {
        try {
            ServerSocket sock = new ServerSocket(4713,100);
            while (true)
                new ClientHandler(sock.accept()).start();
        //It's a blocking method, it makes the program pause and wait until
        //a client connects to the server, then it returns a socket

        }
        catch(IOException e)
            {System.err.println(e);
        }
    }
}
```

1.3 ClientHandler

Denna class hanterar kommunikationen med en klient som ansluter till servern. Klassen ärver från klassen Thread och skapar en ny tråd för varje klient som ansluter till servern.

⟨*ClientHandler*⟩≡
```
class ClientHandler extends Thread {
    static int numberOfPls=0;
    BufferedReader in;
    PrintWriter out;

⟨KonstruktorClientHandler⟩

⟨run()⟩
}
```

1.3.1 Konstruktor

konsruktorn tar emot en socket som parameter och skapar en BufferedReader och en PrintWriter för att kunna läsa och skriva till klienten.

⟨*KonstruktorClientHandler*⟩≡

```
public ClientHandler(Socket socket){
        try {
            in = new BufferedReader(new InputStreamReader
                                    (socket.getInputStream()));
            out= new PrintWriter(socket.getOutputStream());
        }
        catch(IOException e) {System.err.println(e);
        }
    }
```

1.3.2 run

Denna metod är för att starta tråden och hantera kommunikationen med klienten. Klienten skickar sitt namn till servern och servern skickar ett välkomstmeddelande till klienten. Sedan skickar klienten sitt val av hand (STEN, SAX eller PASE) till servern och servern skickar ett slumpmässigt val av hand till klienten. Detta upprepas tills klienten stänger ner anslutningen genom att skicka en tom sträng eller null.

⟨*run()*⟩≡

```
public void run() {
    Random random=new Random();
    String[] hand={"STEN","SAX","PASE"};
    try {
        String name=in.readLine();
        System.out.println((++numberOfPls)+": "+name);
        out.println("Hello, "+name);
        out.flush();
        while(true) {
            String input = in.readLine();
            if(input==null || input.equals("")) break;
            out.println(hand[random.nextInt(3)]);
            out.flush();
        }
        out.println("Bye " + name); out.flush();
        System.out.println(name + " stopped playing");
        numberOfPls--;
    }
    catch(Exception e) {
        System.err.println(e);
    }
}
```

2 RPSModel

Denna klass hanterar kommunikationen med servern och innehåller metoder för att skicka spelarens drag till servern, få serverns drag och avgöra vinnaren. Den skapar ne socket för att ansluta till servern och skapar en BufferedReader och en PrintWriter för att kunna läsa och skriva till servern. De olika metoderna i klassen används för att skicka spelarens drag till servern, få serverns drag och avgöra vinnaren.

⟨*RPSModel.java*⟩≡

```
⟨ImportsRPSModel⟩

public class RPSModel {
    private Socket socket;
    private BufferedReader in;
    private PrintWriter out;

⟨KonstruktorRPSModel⟩

⟨getServerMove()⟩

⟨closeConnection()⟩

⟨determineWinner()⟩
}
```

2.0.1 Imports

Importerar de klasser som behövs för att skapa en socket och kunna läsa och skriva till servern.

⟨*ImportsRPSModel*⟩≡

```
import java.io.*;
import java.net.*;
```

2.0.2 Konstruktor

Konstruktorn skapar en socket för att ansluta till servern och skapar en BufferedReader och en PrintWriter för att kunna läsa och skriva till servern. Socket tar in serverns IP-adress och portnummer som parametrar. Här är IP-adressen ”localhost” för att man kör server på samma dator och portnumret 4713.

⟨*KonstruktorRPSModel*⟩≡

```
public RPSModel() {
    // Anslut till servern
    try {
        socket = new Socket("localhost", 4713);
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        out = new PrintWriter(socket.getOutputStream(), true);

        // Skicka ditt namn till servern
        out.println("Spelare");
        out.flush();

        // Läs serverns hälsning
        String response = in.readLine();
        System.out.println("Servern: " + response);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

2.0.3 getServerMove

⟨*getServerMove()*⟩≡

```
// Skicka spelarens drag till servern och få serverns drag
public String getServerMove(String playerMove) {
    out.println(playerMove);
    out.flush();
    try {
        String serverMove = in.readLine();
        return serverMove;
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}
```

2.0.4 closeConnection

⟨*closeConnection()*⟩≡

```
// Stäng anslutningen
public void closeConnection() {
    try {
        out.println("");
        out.flush();
        in.close();
        out.close();
        socket.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

2.0.5 determineWinner

⟨*determineWinner()*⟩≡

```
// Metod för att avgöra vinnaren
public String[] determineWinner(String playerMove, String serverMove) {
    if (playerMove.equals(serverMove)) {
        return new String[] {"Oavgjort!", "Oavgjort!"};
    } else if ((playerMove.equals("STEN") && serverMove.equals("SAX")) ||
               (playerMove.equals("SAX") && serverMove.equals("PASE")) ||
               (playerMove.equals("PASE") && serverMove.equals("STEN"))) {
        return new String[] {"Du vann!", "Du förlorade!"};
    } else {
        return new String[] {"Du förlorade!", "Du vann!"};
    }
}
```

3 RPSSkel

⟨*RPSSkel.java*⟩≡
⟨*ImportsRPSSkel*⟩

```
public class RPSSkel extends JFrame implements ActionListener {
    Gameboard myboard, computersboard;
    JButton closebutton;
    int counter = 0; // För att räkna ETT... TVÅ... TRE
    RPSModel model;
```

⟨*KonstruktorRPSSkel*⟩

⟨*actionPerfomed()*⟩

```
    public static void main(String[] args) {
        new RPSSkel();
    }
}
```

3.1 Imports

importar de klasser som behövs för att skapa ett grafiskt användargränssnitt.

⟨*ImportsRPSSkel*⟩≡

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.net.*;
import java.io.*;
import java.util.*;
```

3.2 Konstruktor

Konstruktorn skapar ett grafiskt användargränssnitt med två spelplaner, en för spelaren och en för datorn.

⟨*KonstruktorRPSSkel*⟩≡

```
public RPSSkel() {
    super("Malte&Sina"); //sätter titel
    model = new RPSModel();

    setDefaultCloseOperation(EXIT_ON_CLOSE);
    closebutton = new JButton("Avsluta");

    // Skapa spelplaner
    // 'this' är lyssnare för spelknapparna
    myboard = new Gameboard("Sina&Malte", this);
    computersboard = new Gameboard("Datorn");

    // Lägg till spelplanerna i en panel
    JPanel boards = new JPanel();
    boards.setLayout(new GridLayout(1, 2));
    boards.add(myboard);
    boards.add(computersboard);

    // Lägg till lyssnare för Avsluta
    closebutton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            model.closeConnection();
            System.exit(0);
        }
    });

    // Lägg till komponenterna till ramen
    add(boards, BorderLayout.CENTER);
    add(closebutton, BorderLayout.SOUTH);

    setSize(350, 650);
    setVisible(true);
}
```

3.3 actionPerformed

ActionPerformed är en metod som anropas när en händelse inträffar. Den måste vara med när man implementerar ActionListener.

⟨*actionPerfomed()*⟩≡

```
// Implementera actionPerformed för spelknapparna
public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();
    counter++;

    //logiken för spelet och hur man vinner
    if (counter == 1) {
        myboard.resetColor();
        computersboard.resetColor();
        myboard.setLower("ETT...");
        computersboard.setLower("ETT...");
    } else if (counter == 2) {
        myboard.setLower("TVÅ...");
        computersboard.setLower("TVÅ...");
    } else if (counter == 3) {
        // Registrera spelarens drag
        myboard.markPlayed(command);
        myboard.setUpper(command);

        // Skicka spelarens drag till servern och få serverns drag
        String serverMove = model.getServerMove(command);

        // Visa serverns drag
        computersboard.markPlayed(serverMove);
        computersboard.setUpper(serverMove);

        // Avgör vinnaren
        String[] result = model.determineWinner(command, serverMove);

        // Uppdatera resultat
        myboard.setLower(result[0]);
        computersboard.setLower(result[1]);

        // Uppdatera poäng
        if (result[0].equals("Du vann!")) {
            myboard.wins();
        } else if (result[0].equals("Du förlorade!")) {
            computersboard.wins();
        }

        counter = 0; // Återställ räknaren
```

```
        }
    }
```

4 Gameboard

Klassen Gameboard är en JPanel som innehåller 5 JPanels. Första JPanel innehåller spelarens namn och senaste drag som visas i två JLabels. 3 JButtons för att välja sten, sax eller påse. Andra JPanel innehåller meddelanden om spelet och poängen. Sist så finns det en JPanel med 2 JLabels för att visa poängen och spelets status. Klassen innehåller metoder för att sätta texten i JLabels, markera vilken knapp som spelaren valt och uppdatera poängen.

⟨*Gameboard.java*⟩≡
```
⟨ImportsGameboard⟩
class Gameboard extends JPanel {

    private Icon[] icons = {new ImageIcon("rock.gif"),
                    new ImageIcon("paper.gif"),
                    new ImageIcon("scissors.gif")};

    private JButton[] buttons = new JButton[3];
    private JButton lastPlayed; // remembers last chosen button/gesture
    private String[] texts = {"STEN", "PASE", "SAX"};
    private JLabel upperMess, lowerMess, scorelabel;
    private int score;
    private Color bgcolor;
    // HashMap för att koppla gesternas text till motsvarande knapp
    private HashMap<String,JButton> map = new HashMap<String,JButton>();

⟨KonstruktorGameboard⟩

⟨smallMethods⟩

}
```

4.1 Imports

Importerar de klasser som behövs för att skapa en JPanel och de olika komponenterna som behövs för att skapa en spelplan.

⟨*ImportsGameboard*⟩≡
```
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

4.2 Konstruktor

Konstruktorn skapar en JPanel med 5 JPanels. Första JPanel innehåller spelarens namn och senaste drag som visas i två JLabels.

⟨*KonstruktorGameboard*⟩≡
    ⟨*KonstruktorGameboard1*⟩

    ⟨*KonstruktorGameboard2*⟩

4.2.1 Konstruktor 1

konstruktorn skapar en JPanel med 5 JPanels. Första JPanel innehåller spelarens namn och senaste drag som visas i två JLabels.

⟨*KonstruktorGameboard1*⟩≡

```
// Constructor that builds the board, used for computers board
Gameboard(String name) {
            setLayout(new GridLayout(5,1));

            // Upper JPanel holds players name and last gesture played
            JPanel upper = new JPanel();
            upper.setLayout(new GridLayout(2,1));
            upper.add(new JLabel(name, JLabel.CENTER));
            upperMess = new JLabel("RPS", JLabel.CENTER);
            upper.add(upperMess);
            add(upper);

            // Lower JPanel has messages about the game and score
            JPanel lower = new JPanel();
            lower.setLayout(new GridLayout(2,1));
            lowerMess = new JLabel("börja spela", JLabel.CENTER);
            scorelabel = new JLabel("Score: 0", JLabel.CENTER);
            lower.add(lowerMess); lower.add(scorelabel);

            for (int i = 0; i<3; i++){
                    buttons[i] = new JButton(icons[i]);
        //sätter actionCommand för varje knapp till sten sax pase
                    buttons[i].setActionCommand(texts[i]);
                    add(buttons[i]);
                    // Store each button in a map with its text as key.
                    // Enables us to retrieve the button from a textvalue.
                    map.put(texts[i],buttons[i]);
            }

            add(lower); // added after buttons
            bgcolor = buttons[0].getBackground();
            lastPlayed = buttons[0]; // arbitrary value at start
}
```

4.2.2 Konstruktor 2

konstruktorn skapar en JPanel med 5 JPanels. Första JPanel innehåller spelarens namn och senaste drag som visas i två JLabels.

⟨*KonstruktorGameboard2*⟩≡

```
// Contructor for players board, puts listener on buttons
Gameboard(String name, ActionListener listener) {
            this(name); // call other constructor to build the board

    // Lägg till action-lyssnaren till varje knapp
    // så att de kan reagera på klick
            for (int i = 0; i<3; i++)
                    buttons[i].addActionListener(listener);
}
```

4.3 Små metoder

Dessa metoder är för att sätta texten i JLabels, markera vilken knapp som spelaren valt och uppdatera poängen. resetColor() sätter tillbaka färgen på knappen till dess ursprungliga färg. setUpper() sätter texten i JLabels för spelarens senaste drag. setLower() sätter texten i JLabels för meddelanden om spelet. markPlayed() sätter färgen på knappen till gul för att markera att spelaren har valt den. Du antingen skicka in en sträng eller en JButton som parameter. wins() ökar poängen med 1 och uppdaterar poängen.

⟨*smallMethods*⟩≡

```
// reset yellow color
void resetColor() {
            lastPlayed.setBackground(bgcolor);
}

void setUpper(String r) {
            upperMess.setText(r);
}

void setLower(String r) {
            lowerMess.setText(r);
}

// remember last chosen JButton and mark it yellow
void markPlayed(String r) {
            lastPlayed = map.get(r);
            lastPlayed.setBackground(Color.yellow);
}

// or use JButton as parameter
void markPlayed(JButton b) {
            lastPlayed = b;
            lastPlayed.setBackground(Color.yellow);
}

// add one point and display new score
void wins() {
            score++;
            scorelabel.setText("Score: " + score);
}
```