НОВ БЪЛГАРСКИ УНИВЕРСИТЕТ

## МАГИСТЪРСКИ ФАКУЛТЕТ

# ДЕПАРТАМЕНТ "ИНФОРМАТИКА"

**ПРОГРАМА „Софтуерни технологии в Интернет”**



### Курсов проект

**INFM169 Проект по обектноориентирано програмиране в Интернет**

**ТЕМА: Фризьорски салон**

**НА СТУДЕНТА** : Георги Иванов Георгиев

**Фак. №:** F88683

**Дата:** 06/02/2018 г.

**Преподавател:**

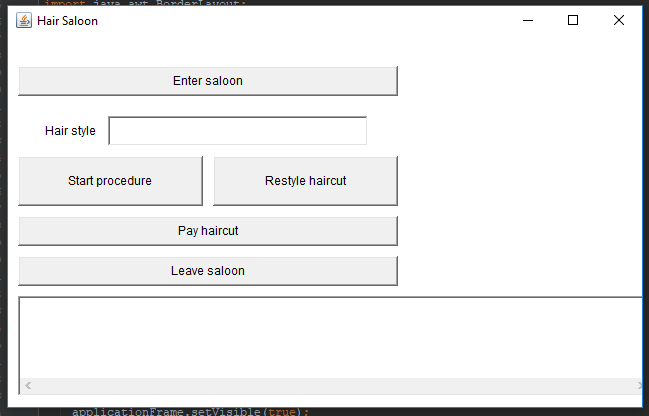
/ проф. д-р. И.Момчев / …………………

София

1. Въведение

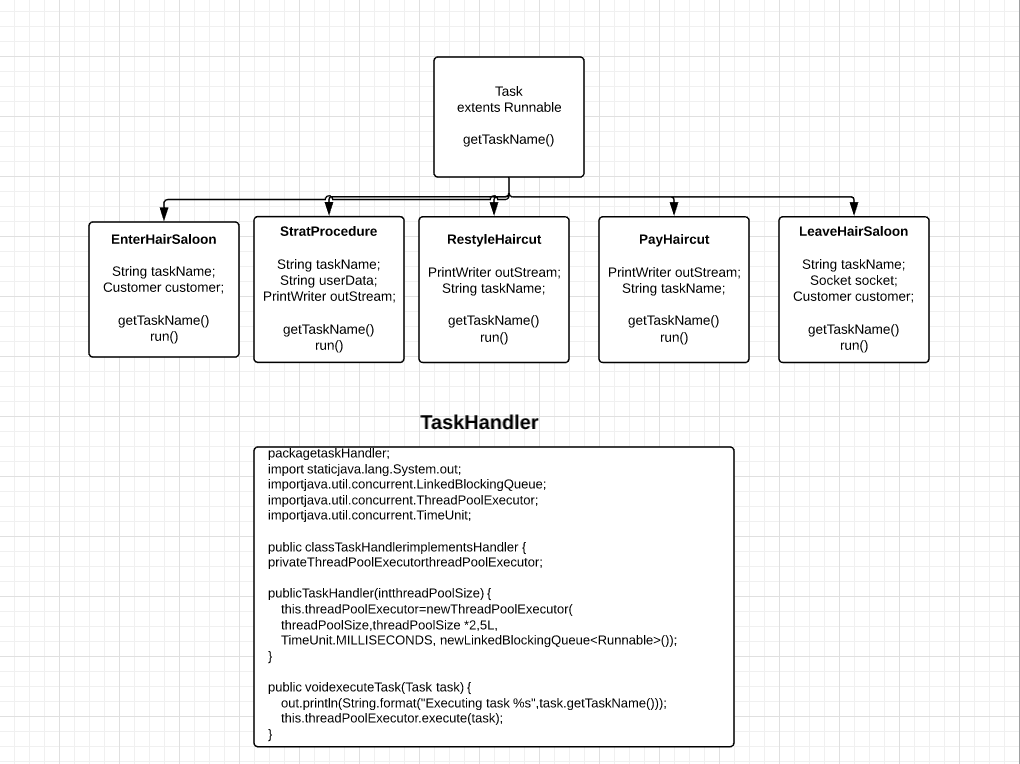
Курсовата работа „Фризьорски салон“ представлява приложение демонстриращо работата на клиент-сървър постановката с реализация на вътрешен оркестратор на задачи и борба за ресурси. Клиентът е поставен в позиция да избира вход в салона и по този начин за него започва работата на приложението. Изборът на бутона вход генерира задача, която се поема от оркестратора и се изпълнява при първа възможност. Веднъж влязъл клиентът въвежда вид фризура в предназначеното за това текстово поле. Това отново генерира задача и тя отново се изпълнява от оркестратора. Информацията за избраната фризура преминава през сокета отворен от сървъра и ако стринга дефиниращ вида фризура не е празен, задачата се изпълнява определено случайно избрано време. След изпълнение на първоначалната фризура, клиентът е поставен през избора да оправи прическата или да я приеме и да плати. Изборът на бутона за повторно стилизиране води до изпълнение на нова задача случайно генерирано време. Изборът на бутона за сметка генерира задача, както и изборът на бутона за изход. Специфичното при изход е освобождаване на мястото заето от клиента.

1. Представяне на потребителския интерфейс



Вече беше споменато, че всеки бутон от потребителския UI генерира задача представена чрез обект имплементиращ интерфейса Task.

Йерархията на класовете имплементирали интерфейса е следната:



TaskHandler e реализация на threadPool, който приема за изпълнение задача от тип Task и я изпълнява, когато има свободна нишка за това.

В текстовото поле се изписват съобщения за потребителя използващ графичния интерфейс. Възможните съобщения са:

* Client is waiting to enter the saloon– клиентът чака да се освободи място в салона. Задачата за вход се изпълнява от оркестратора, но няма свободни места.
* RealCustomer entered the saloon– задачата за вход е приключила изпълнение.
* Please describe your desired haircut– клиентът не е посочил валиден стринг за избрана прическа.
* Working on the haircut – задачата за прическа се изпълнява.
* Client haircut style is: " + userOder + " If you like your hairstyle you can proceed to payment. Else you can restyle your hair.– задачата за изпълнение на прическа е приключила. Потребителят може да рестилизира или да плати.
* Restyling haircut – задачата за рестилизиране се изпълнява.
* Customer haircut has been restyled. Please pay your service. – задачата за рестилизиране е приключила. Клиентът следва да плати сметка.
* Waiting to pay for service. – клиентът изчаква освобождаване на нишка за изпълнение на задача по плащане.
* Customer has paid for the haircut – задачата за плащане е приключила. Клиентът следва да напусне салона.
* Leaving saloon. – клиентът е започнал задача по напускане.
* RealCustomer left the saloon – клиентът е напуснал салона и е освободил място.

1. Описание на класове
   1. Пакет app – съдържа клас HairSaloonApplication, който стартира клиентското приложение.
   2. Пакет common – съдържа класове Customer (обособява наличието на клиент на сървъра) и Entrance (съдържа основни задачи свързани с ресурса място в салона).
   3. Пакет hairSaloon – съдържа клас HairSaloonServer, който стартира сървъра.
   4. Пакет taskHandler – съръжа класове EnterHairSaloonTask, LeaveHairSaloonTask, PayHaircutTask, RestyleHaircutTask, StartProcedureTask, TaskHandler и интерфейсите Handler и Task.
   5. Пакет ui – съдържа класовете EnterL, HairSaloonApplet, LeaveSaloonL, PayHaircutL, RestyleHairCutL и StartProcedureL.
2. Текст на програмата – може да бъде открит на адрес: <https://github.com/Georgegig/nbu-hair-saloon>

package app;  
  
import ui.HairSaloonApplet;  
  
public class HairSaloonApplication {  
 public static void main(String[] args) {  
 HairSaloonApplet hairSaloonApplet = new HairSaloonApplet();  
 hairSaloonApplet.init();  
 hairSaloonApplet.start();  
 }  
}

package common;  
import java.io.PrintWriter;  
  
public class Customer {  
 private Entrance entrance;  
 private String type;  
 private PrintWriter outStream;  
  
 public Customer(Entrance entrance, String type, PrintWriter outStream) {  
 this.entrance = entrance;  
 this.type = type;  
 this.outStream = outStream;  
 }  
  
 public void takeSeat() {  
 this.entrance.takeSeat(type, outStream);  
 }  
  
 public void freeSeat() {  
 this.entrance.freeSeat(type, outStream);  
 }  
}

package common;  
import static java.lang.System.*out*;  
import java.io.PrintWriter;  
  
public class Entrance {  
 private int seatCount;  
  
 public Entrance(int seatCount){  
 this.seatCount = seatCount;  
 }  
  
 public synchronized void takeSeat(String type, PrintWriter outStream) {  
 while (seatCount == 0) {  
 printMessage("log", outStream, " waiting for a free seat");  
 try {  
 wait();  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 *out*.println(e.getMessage());  
 }  
 }  
 seatCount--;  
 *out*.println(String.*format*("Seats in the saloon after taking one: %s", seatCount));  
 printMessage(type, outStream, " entered the saloon");  
 }  
  
 public synchronized void freeSeat(String type, PrintWriter outStream) {  
 seatCount++;  
 *out*.println(String.*format*("Seats in the saloon after freeing one: %s", seatCount));  
 printMessage(type, outStream, " left the saloon");  
 notifyAll();  
 }  
  
 private void printMessage(String type, PrintWriter outStream, String message) {  
 if ("RealCustomer".equals(type)) {  
 outStream.println(type + message);  
 *out*.println(type + message);  
 } else {  
 *out*.println(Thread.*currentThread*().getName() + message);  
 }  
 }  
}

package hairSaloon;  
import common.Customer;  
import common.Entrance;  
import taskHandler.\*;  
import static java.lang.System.\*;  
import java.io.\*;  
import java.net.ServerSocket;  
import java.net.Socket;  
  
public class HairSaloonServer {  
  
 public static void main(String[] args) throws Exception {  
 *out*.println("The hair saloon server is running.");  
 ServerSocket serverSocket = new ServerSocket(5400);  
 TaskHandler taskHandler = new TaskHandler(10);  
 Entrance entrance = new Entrance(2);  
 try {  
 while (true) {  
 new Order(serverSocket.accept(), taskHandler, entrance).start();  
 }  
 } catch (Exception exc) {  
 *out*.println("Error! - " + exc.toString());  
 }  
 }  
  
 private static class Order extends Thread {  
 private Socket socket;  
 private TaskHandler taskHandler;  
 private Entrance entrance;  
  
 Order(Socket socket, TaskHandler taskHandler, Entrance entrance) {  
 this.socket = socket;  
 this.taskHandler = taskHandler;  
 this.entrance = entrance;  
 }  
  
 // run method implemented by Thread class  
 @Override  
 public void run() {  
 try {  
 BufferedReader dataInputStream = new BufferedReader(  
 new InputStreamReader(socket.getInputStream()));  
  
 PrintWriter printStream = new PrintWriter(new BufferedWriter(  
 new OutputStreamWriter(socket.getOutputStream())),true);  
  
 while (true) {  
 *out*.println("Debug: Wait for client request");  
 String request = dataInputStream.readLine();  
 *out*.println("Debug: " + request);  
  
 Customer customer = new Customer(this.entrance, "RealCustomer", printStream);  
  
 if ("enter".equals(request)) {  
 EnterHairSaloonTask enterTask = new EnterHairSaloonTask("RealCustomerEnterTask", customer);  
 taskHandler.executeTask(enterTask);  
 }  
 else if (request.contains("procedure:")){  
 StartProcedureTask startProcedureTask = new StartProcedureTask("RealCustomerProcedureTask",  
 request, printStream);  
 taskHandler.executeTask(startProcedureTask);  
 }  
 else if ("restyle".equals(request)){  
 RestyleHaircutTask restyleHaircutTask = new RestyleHaircutTask("RealCustomerRestyleTask", printStream);  
 taskHandler.executeTask(restyleHaircutTask);  
 }  
 else if ("pay".equals(request)){  
 PayHaircutTask payHaircutTask = new PayHaircutTask("RealCustomerPayTask", printStream);  
 taskHandler.executeTask(payHaircutTask);  
 }  
 else if ("leave".equals(request)){  
 LeaveHairSaloonTask leaveTask = new LeaveHairSaloonTask("RealCustomerLeaveTask", socket, customer);  
 taskHandler.executeTask(leaveTask);  
 }  
 }  
 } catch (NullPointerException exc) {  
 *out*.println("Client has closed the connection!");  
 } catch (Exception exc) {  
 *out*.println("Error! - " + exc.toString());  
 }  
  
 try {  
 socket.close();  
 } catch (Exception exc) {  
 *out*.println("Error! - " + exc.toString());  
 }  
 }  
 }  
}

package taskHandler;  
import common.Customer;  
import static java.lang.System.*out*;  
  
public class EnterHairSaloonTask implements Task {  
 private String taskName;  
 private Customer customer;  
  
 public EnterHairSaloonTask(String taskName, Customer customer) {  
 this.taskName = taskName;  
 this.customer = customer;  
 }  
  
 public void run() {  
 try {  
 this.customer.takeSeat();  
 } catch (Exception exc) {  
 *out*.println("Error in entering the hair saloon - " + exc.toString());  
 }  
 }  
  
 public String getTaskName() {  
 return this.taskName;  
 }  
}

package taskHandler;  
  
public interface Handler {  
  
 void executeTask(Task task);  
}

package taskHandler;  
import static java.lang.System.*out*;  
import common.Customer;  
import java.net.Socket;  
  
public class LeaveHairSaloonTask implements Task {  
 private String taskName;  
 private Socket socket;  
 private Customer customer;  
  
 public LeaveHairSaloonTask(String taskName, Socket socket, Customer customer) {  
 this.taskName = taskName;  
 this.socket = socket;  
 this.customer = customer;  
 }  
  
 public void run() {  
 this.customer.freeSeat();  
 try {  
 socket.close();  
 } catch (Exception exc) {  
 *out*.println("Error in close the socket - " + exc.toString());  
 }  
 }  
  
 public String getTaskName() {  
 return taskName;  
 }  
}

package taskHandler;  
import java.io.PrintWriter;  
  
public class PayHaircutTask implements Task {  
 private PrintWriter outStream;  
 private String taskName;  
  
 public PayHaircutTask(String taskName, PrintWriter outStream) {  
 this.taskName = taskName;  
 this.outStream = outStream;  
 }  
  
 public void run() {  
 outStream.println("Customer has paid for the haircut");  
 }  
  
 public String getTaskName() {  
 return taskName;  
 }  
}

package taskHandler;  
import java.io.PrintWriter;  
import static java.lang.System.*out*;  
import static java.lang.Thread.*sleep*;  
  
public class RestyleHaircutTask implements Task {  
 private PrintWriter outStream;  
 private String taskName;  
  
 public RestyleHaircutTask(String taskName, PrintWriter outStream){  
 this.taskName = taskName;  
 this.outStream = outStream;  
 }  
  
 public String getTaskName() {  
 return this.taskName;  
 }  
  
 public void run() {  
 try {  
 *sleep*((int)(Math.*random*()\*1000));  
 } catch (InterruptedException e){  
 *out*.println(String.*format*("Failed executing task: %s", getTaskName()));  
 }  
 outStream.println("Customer haircut has been restyled. Please pay your service.");  
 }  
}

package taskHandler;  
import java.io.PrintWriter;  
import java.util.Arrays;  
import java.util.List;  
  
import static java.lang.System.*out*;  
import static java.lang.Thread.*sleep*;  
  
public class StartProcedureTask implements Task {  
 private String taskName;  
 private String userData;  
 private PrintWriter outStream;  
  
 public StartProcedureTask(String taskName, String userData, PrintWriter outStream) {  
 this.taskName = taskName;  
 this.userData = userData;  
 this.outStream = outStream;  
 }  
  
 public void run() {  
 List<String> userDataList = Arrays.*asList*(userData.split(":"));  
 String userOder = "";  
 try {  
 userOder += userDataList.get(1);  
 } catch (IndexOutOfBoundsException e) {  
 outStream.println("Please describe your desired haircut");  
 return;  
 }  
 try {  
 *sleep*((int)(Math.*random*()\*1000));  
 } catch (InterruptedException e){  
 *out*.println(String.*format*("Failed executing task: %s", getTaskName()));  
 }  
  
 outStream.println("Client haircut style is: " + userOder + " If you like your hairstyle you can proceed to payment. Else you can restyle your hair.");  
 }  
  
 public String getTaskName() {  
 return this.taskName;  
 }  
}

package taskHandler;  
  
interface Task extends Runnable {  
 String getTaskName();  
}

package taskHandler;  
import static java.lang.System.*out*;  
import java.util.concurrent.LinkedBlockingQueue;  
import java.util.concurrent.ThreadPoolExecutor;  
import java.util.concurrent.TimeUnit;  
  
public class TaskHandler implements Handler {  
 private ThreadPoolExecutor threadPoolExecutor;  
  
 public TaskHandler(int threadPoolSize) {  
 this.threadPoolExecutor = new ThreadPoolExecutor(  
 threadPoolSize, threadPoolSize \* 2, 5L,  
 TimeUnit.*MILLISECONDS*, new LinkedBlockingQueue<Runnable>());  
 }  
  
 public void executeTask(Task task) {  
 *out*.println(String.*format*("Executing task %s", task.getTaskName()));  
 this.threadPoolExecutor.execute(task);  
 }  
}

package ui;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.PrintWriter;  
  
public class EnterL implements ActionListener {  
 private PrintWriter outStream;  
 private BufferedReader inStream;  
 private TextArea outputArea;  
  
 EnterL(PrintWriter outStream, BufferedReader inStream, TextArea outputArea) {  
 this.outStream = outStream;  
 this.inStream = inStream;  
 this.outputArea = outputArea;  
 }  
  
 public void actionPerformed(ActionEvent e) {  
 outputArea.setText("Client is waiting to enter the saloon");  
 outStream.println("enter");  
 try {  
 String serverResponse = inStream.readLine();  
 outputArea.setText(serverResponse);  
 } catch (IOException e1) {  
 outputArea.setText("Something went wrong when trying to enter the hair saloon");  
 e1.printStackTrace();  
 }  
 }  
}

package ui;  
import static java.lang.System.*exit*;  
import static java.lang.System.*out*;  
import java.applet.Applet;  
import java.awt.BorderLayout;  
import java.awt.Button;  
import java.awt.Frame;  
import java.awt.Label;  
import java.awt.TextArea;  
import java.awt.TextField;  
import java.awt.event.WindowAdapter;  
import java.awt.event.WindowEvent;  
import java.io.\*;  
import java.net.Socket;  
  
  
public class HairSaloonApplet extends Applet{  
 private BufferedReader inStream;  
 private PrintWriter outStream;  
  
 @Override  
 public void init() {  
 Frame applicationFrame = new Frame("Hair Saloon");  
 applicationFrame.addWindowListener(  
 new WindowAdapter() {  
 @Override  
 public void windowClosing(WindowEvent e) {  
 *exit*(0);  
 }  
 }  
 );  
 applicationFrame.add(this, BorderLayout.*CENTER*);  
 applicationFrame.setSize(650, 410);  
 applicationFrame.setVisible(true);  
 setLayout(null);  
  
 try {  
 Socket socket = new Socket("localhost", 5400);  
 inStream = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 outStream = new PrintWriter(new BufferedWriter(  
 new OutputStreamWriter(socket.getOutputStream())),true);  
 } catch (Exception exc) {  
 *out*.println("Error in initialize the network - " + exc.toString());  
 }  
  
 TextArea outputArea = new TextArea();  
 outputArea.setBounds(10, 260, 650, 100);  
 add(outputArea);  
  
 Label hairStyleLabel = new Label("Hair style", Label.*RIGHT*);  
 hairStyleLabel.setBounds(10, 80, 80, 30);  
 add(hairStyleLabel);  
  
 TextField hairStyleField = new TextField();  
 hairStyleField.setBounds(100, 80, 260, 30);  
 add(hairStyleField);  
  
 Button enterSaloonButton = new Button("Enter saloon");  
 enterSaloonButton.setBounds(10, 30, 380, 30);  
 enterSaloonButton.addActionListener(new EnterL(outStream, inStream, outputArea));  
 add(enterSaloonButton);  
  
 Button startProcedureButton = new Button("Start procedure");  
 startProcedureButton.setBounds(10, 120, 185, 50);  
 startProcedureButton.addActionListener(  
 new StartProcedureL(outStream, inStream, outputArea, hairStyleField));  
 add(startProcedureButton);  
  
 Button restyleHaircutButton = new Button("Restyle haircut");  
 restyleHaircutButton.setBounds(205, 120, 185, 50);  
 restyleHaircutButton.addActionListener(new RestyleHairCutL(outStream, inStream, outputArea));  
 add(restyleHaircutButton);  
  
 Button payHaircutButton = new Button("Pay haircut");  
 payHaircutButton.setBounds(10, 180, 380, 30);  
 payHaircutButton.addActionListener(new PayHaircutL(outStream, inStream, outputArea));  
 add(payHaircutButton);  
  
 Button exitSaloonButton = new Button("Leave saloon");  
 exitSaloonButton.setBounds(10, 220, 380, 30);  
 exitSaloonButton.addActionListener(new LeaveSaloonL(outStream, inStream, outputArea));  
 add(exitSaloonButton);  
  
 }  
}

package ui;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.PrintWriter;  
  
public class LeaveSaloonL implements ActionListener {  
 private PrintWriter outStream;  
 private BufferedReader inStream;  
 private TextArea outputArea;  
  
 LeaveSaloonL(PrintWriter outStream, BufferedReader inStream, TextArea outputArea) {  
 this.outStream = outStream;  
 this.inStream = inStream;  
 this.outputArea = outputArea;  
 }  
  
 public void actionPerformed(ActionEvent e) {  
 outputArea.setText("Leaving saloon.");  
 outStream.println("leave");  
 try {  
 String serverResponse = inStream.readLine();  
 outputArea.setText(serverResponse);  
 System.*exit*(0);  
 } catch (IOException e1) {  
 outputArea.setText("Something went wrong when trying to leave the hair saloon");  
 e1.printStackTrace();  
 }  
 }  
}

package ui;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.PrintWriter;  
  
public class PayHaircutL implements ActionListener {  
 private PrintWriter outStream;  
 private BufferedReader inStream;  
 private TextArea outputArea;  
  
 PayHaircutL(PrintWriter outStream, BufferedReader inStream, TextArea outputArea){  
 this.outStream = outStream;  
 this.inStream = inStream;  
 this.outputArea = outputArea;  
 }  
  
 public void actionPerformed(ActionEvent e) {  
 outputArea.setText("Waiting to pay for service.");  
 outStream.println("pay");  
 try {  
 String serverResponse;  
 while (true) {  
 if (inStream.ready()) {  
 serverResponse = inStream.readLine();  
 break;  
 }  
 }  
 outputArea.setText(serverResponse);  
 } catch (IOException e1) {  
 outputArea.setText("Something went wrong when trying to pay for the haircut");  
 e1.printStackTrace();  
 }  
 }  
}

package ui;  
import java.awt.TextArea;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.PrintWriter;  
  
public class RestyleHairCutL implements ActionListener {  
 private PrintWriter outStream;  
 private BufferedReader inStream;  
 private TextArea outputArea;  
  
  
 RestyleHairCutL(PrintWriter outStream, BufferedReader inStream, TextArea outputArea) {  
 this.outStream = outStream;  
 this.inStream = inStream;  
 this.outputArea = outputArea;  
 }  
  
 public void actionPerformed(ActionEvent e) {  
 outputArea.setText("Restyling haircut");  
 outStream.println("restyle");  
 try {  
 String serverResponse;  
 while (true) {  
 if (inStream.ready()) {  
 serverResponse = inStream.readLine();  
 break;  
 }  
 }  
 outputArea.setText(serverResponse);  
 } catch (IOException e1) {  
 outputArea.setText("Something went wrong when trying to restyle the haircut");  
 e1.printStackTrace();  
 }  
 }  
}

package ui;  
import java.awt.TextArea;  
import java.awt.TextField;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.PrintWriter;  
  
public class StartProcedureL implements ActionListener {  
 private PrintWriter outStream;  
 private BufferedReader inStream;  
 private TextArea outputArea;  
 private TextField hairStyleTextField;  
  
 StartProcedureL(PrintWriter outStream, BufferedReader inStream, TextArea outputArea, TextField hairStyleTextField ) {  
 this.outStream = outStream;  
 this.inStream = inStream;  
 this.outputArea = outputArea;  
 this.hairStyleTextField = hairStyleTextField;  
 }  
  
 public void actionPerformed(ActionEvent e) {  
 outStream.println(String.*format*("procedure:%s", hairStyleTextField.getText()));  
 outputArea.setText("Working on the haircut");  
 String serverResponse;  
 try {  
 while (true) {  
 if (inStream.ready()) {  
 serverResponse = inStream.readLine();  
 break;  
 }  
 }  
 outputArea.setText(serverResponse);  
 } catch (IOException e1) {  
 outputArea.setText("Something went wrong when trying get haircut in the saloon");  
 e1.printStackTrace();  
 }  
 }  
}

1. Инструкция за инсталация

Source кодът се сваля от горепосоченият адрес. При наличието на подходяща среда за разработка (IDE) се стартира файл HairSaloonServer и след това желан на брой пъти се стартира файл HairSaloonApplication.

1. Инструкция за експлоатация

Потребителското поведение трябва да следва следната последователност:

1. Потребителят натиска бутона за влизане в салона и изчаква своя ред;
2. Потребителят въвежда желаната прическа и избира бутона за започване на процедура;
3. Потребителя преценява дали резултата от прическата е задоволителен. Ако е натиска бутона плати сметка. Ако не е натиска бутона рестилизирай;
4. В случай на рестилизиране потребителя изчаква то да приключи, след което заплаща своята сметка;
5. Потребителят натиска бутона напусни салона и напуска салона. Сокет връзката се прекратява и клиентското приложение се затваря.
6. Защо да се използва приложението

Приложението демонстрира базова клиент-сървър постановката. Графичният интерфейс е лесен за реализация. Изпълнението на задачите генерирани от бутоните осъществява идеята на threadPool execution. Условието за борба за място в салона от друга страна илюстрира работата със синхронизиране на нишки. Простотата на приложението би позволила лесното навлизане в основите на многонишковото и клиент-сървърното програмиране с java.