ChatServer.java

-------------------------------------------------------------------------------------

package serverChat;

import java.io.\*;

import java.lang.reflect.InvocationTargetException;

import java.lang.reflect.Method;

import java.net.\*;

public class ChatServer {

private static int serverPort;

private Class<?> mainClass; // Ссылка на класс Main

public ChatServer(int serverPort)

{

this.serverPort = serverPort;

}

public void setMainClass(Class<?> mainClass) {

this.mainClass = mainClass;

}

public void ChatServ() {

try {

// Создание серверного сокета

ServerSocket serverSocket = new ServerSocket(serverPort);

System.out.println("Сервер запущен. Ожидание подключения клиента...");

// Ожидание подключения клиента

Socket clientSocket = serverSocket.accept();

System.out.println("Клиент подключен.");

// Получение входного и выходного потоков

BufferedReader inputReader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

PrintWriter outputWriter = new PrintWriter(clientSocket.getOutputStream(), true);

// Создание потока для чтения сообщений от клиента

Thread readThread = new Thread(new Runnable() {

@Override

public void run() {

try {

String message;

while ((message = inputReader.readLine()) != null) {

System.out.println("Клиент: " + message);

outputWriter.println("Получено сообщение: " + message);

}

inputReader.close();

outputWriter.close();

} catch (IOException e) {

e.printStackTrace();

}

}

});

readThread.start();

// Отправка сообщений клиенту

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

String serverMessage;

while (true) {

serverMessage = serverReader.readLine();

//Отправка сообщения клиенту

outputWriter.println(serverMessage);

if (serverMessage == null || serverMessage.equalsIgnoreCase("exit")) {

break;

}

}

// Закрытие потоков и сокета

serverReader.close();

inputReader.close();

outputWriter.close();

clientSocket.close();

serverSocket.close();

exitProgram();

} catch (IOException ex) {

ex.printStackTrace();

}

}

private void exitProgram() {

if (mainClass != null) {

try {

// Вызываем метод main() в основном классе для завершения программы

Method mainMethod = mainClass.getMethod("main", String[].class);

mainMethod.invoke(null, (Object) new String[0]);

} catch (NoSuchMethodException | IllegalAccessException | InvocationTargetException e) {

e.printStackTrace();

}

}

}

}} FileServer.java

--------------------------------------------------------------------------------------------------------------------------------------

package serverFile;

import java.io.\*;

import java.lang.reflect.InvocationTargetException;

import java.lang.reflect.Method;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.ArrayList;

import java.util.List;

public class FileServer {

private static int serverPort; // Порт сервера

private static volatile boolean isRunning = true;

private static Class<?> mainClass; // Ссылка на класс Main

private static final String SAVE\_FOLDER = "server\_files"; // Папка для сохранения файлов на сервере

public FileServer(int serverPort,Class<?> mainClass)

{

this.mainClass = mainClass;

this.serverPort = serverPort;

}

public static void Accept() {

try {

ServerSocket serverSocket = new ServerSocket(serverPort);

System.out.println("Сервер запущен. Ожидание подключений...");

while (isRunning) {

Socket clientSocket = serverSocket.accept();

System.out.println("Подключение установлено: " + clientSocket);

// Создаем новый поток для каждого подключения клиента

Thread clientThread = new Thread(new ClientHandler(clientSocket));

clientThread.start();

}

serverSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

private static class ClientHandler implements Runnable {

private Socket clientSocket;

public ClientHandler(Socket clientSocket) {

this.clientSocket = clientSocket;

}

@Override

public void run() {

try {

// Получаем входной поток для чтения данных от клиента

InputStream inputStream = clientSocket.getInputStream();

DataInputStream dataInputStream = new DataInputStream(inputStream);

// Получаем выходной поток для отправки данных клиенту

OutputStream outputStream = clientSocket.getOutputStream();

DataOutputStream dataOutputStream = new DataOutputStream(outputStream);

// Читаем команду от клиента

String command = dataInputStream.readUTF();

System.out.println("Команда от клиента: " + command);

if (command.equals("UPLOAD")) {

// Команда от клиента: UPLOAD - загрузить файл на сервер

// Читаем имя файла от клиента

String fileName = dataInputStream.readUTF();

// Создаем новый файл для сохранения на сервере

File savedFile = new File(SAVE\_FOLDER, fileName);

// Читаем данные файла от клиента и сохраняем на сервере

FileOutputStream fileOutputStream = new FileOutputStream(savedFile);

byte[] buffer = new byte[4096];

int bytesRead;

while ((bytesRead = dataInputStream.read(buffer)) != -1) {

fileOutputStream.write(buffer, 0, bytesRead);

}

System.out.println("Файл сохранен на сервере: " + savedFile.getAbsolutePath());

// Закрываем потоки

fileOutputStream.close();

clientSocket.close();

} else if (command.equals("DOWNLOAD")) {

// Команда от клиента: DOWNLOAD - скачать файл с сервера

// Читаем имя файла от клиента

String fileName = dataInputStream.readUTF();

// Проверяем, существует ли файл на сервере

File file = new File(SAVE\_FOLDER, fileName);

if (file.exists()) {

// Отправляем имя файла клиенту

dataOutputStream.writeUTF(fileName);

// Читаем данные файла и отправляем клиенту

FileInputStream fileInputStream = new FileInputStream(file);

byte[] buffer = new byte[4096];

int bytesRead;

while ((bytesRead = fileInputStream.read(buffer)) != -1) {

dataOutputStream.write(buffer, 0, bytesRead);

}

System.out.println("Файл отправлен клиенту: " + file.getAbsolutePath());

// Закрываем потоки

fileInputStream.close();

clientSocket.close();

} else {

// Отправляем пустое имя файла, чтобы клиент знал, что файл не найден

dataOutputStream.writeUTF("");

System.out.println("Файл не найден на сервере.");

}

} else if (command.equals("VIEW")) {

// Команда от клиента: VIEW - просмотреть файлы на сервере

// Получаем список файлов на сервере

List<String> fileNames = getFileNamesOnServer();

// Отправляем количество файлов клиенту

dataOutputStream.writeInt(fileNames.size());

// Отправляем имена файлов клиенту

for (String fileName : fileNames) {

dataOutputStream.writeUTF(fileName);

}

clientSocket.close();

}else if (command.equals("EXIT")){

isRunning = false;

// Закрываем потоки

dataInputStream.close();

inputStream.close();

dataOutputStream.close();

outputStream.close();

clientSocket.close();

exitProgram();

}

} catch (IOException e) {

e.printStackTrace();

}

}

private void exitProgram() {

try {

// Вызываем метод main() в основном классе для завершения программы

Method mainMethod = mainClass.getMethod("main", String[].class);

mainMethod.invoke(null, new Object[] { new String[0] });

} catch (NoSuchMethodException | IllegalAccessException | InvocationTargetException e) {

e.printStackTrace();

}

}

private List<String> getFileNamesOnServer() {

List<String> fileNames = new ArrayList<>();

File folder = new File(SAVE\_FOLDER);

File[] files = folder.listFiles();

if (files != null) {

for (File file : files) {

if (file.isFile()) {

fileNames.add(file.getName());

}

}

}

return fileNames;

}

}

}ServerIPAddress.java

--------------------------------------------------------------------------------------------------------------

package util;

import java.net.\*;

public class ServerIPAddress {

public static void ipAddress() {

try {

// Получение локального IP-адреса сервера

InetAddress localhost = InetAddress.getLocalHost();

String ipAddress = localhost.getHostAddress();

System.out.println("IP-адрес сервера: " + ipAddress);

} catch (UnknownHostException e) {

e.printStackTrace();

}

}

}

Main.java

import serverChat.ChatServer;

import serverFile.FileServer;

import util.ServerIPAddress;

import java.util.Scanner;

public class Main {

private int serverPort;

private String command;

private static volatile boolean isRunning = true;

private static String Menu = "Приветствую клиент, при помощи команд вы сможете (только для начала пожалуйста введите какой порт использовать серверу и к какому Ip-адресу подключаться)\n" +

"File - отправить файл на сервер\n" +

"Chat - начать диалог с сервером\n" +

"Quit - завершить программу";

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

ServerIPAddress ipAdd = new ServerIPAddress();

System.out.println(Menu);

ipAdd.ipAddress();

System.out.println("Введите порт сервера : ");

int serverPort = scanner.nextInt();

while (isRunning) {

System.out.println("Введите что вы хотите сделать : ");

String command = scanner.next();

switch (command) {

case "File":

System.out.println("Начинаем отправку файлов...");

FileServer client = new FileServer(serverPort,Main.class);

client.Accept();

break;

case "Chat":

System.out.println("Начинаем диалог...");

ChatServer chat = new ChatServer(serverPort);

chat.setMainClass(Main.class);

chat.ChatServ();

break;

case "Quit":

System.out.println("Выполнение программы завершено");

isRunning = false;

break;

default:

System.out.println("Введена неверная команда");

}

}

}

}ChatClient.java

---------------------------------------------------------------------------------------------------------------------------------------

package clientChat;

import java.io.\*;

import java.lang.reflect.InvocationTargetException;

import java.lang.reflect.Method;

import java.net.Socket;

public class ChatClient {

private Class<?> mainClass; // Ссылка на класс Main

private int serverport;

private String serverIp; // Укажите IP-адрес сервера

public ChatClient(String serverIp, int serverport) {

this.serverIp = serverIp;

this.serverport = serverport;

}

public void setMainClass(Class<?> mainClass) {

this.mainClass = mainClass;

}

public void chat() {

try {

// Создание сокета клиента и подключение к серверу

Socket clientSocket = new Socket(serverIp, serverport);

System.out.println("Успешное подключение к серверу.");

// Получение входного и выходного потоков

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

BufferedReader inputReader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

PrintWriter outputWriter = new PrintWriter(clientSocket.getOutputStream(), true);

// Создание потока для чтения сообщений от сервера

Thread readThread = new Thread(() -> {

try {

String response;

while ((response = inputReader.readLine()) != null) {

System.out.println("Сервер: " + response);

}

} catch (IOException e) {

e.printStackTrace();

}

});

readThread.start();

// Отправка сообщений серверу

String message;

while (true) {

message = reader.readLine();

// Отправка сообщения на сервер

outputWriter.println(message);

if (message == null || message.equalsIgnoreCase("exit")) {

System.out.println("Выходим из чата!");

break;

}

}

// Закрытие потоков и сокета

reader.close();

inputReader.close();

outputWriter.close();

clientSocket.close();

exitProgram();

} catch (IOException ex) {

ex.printStackTrace();

}

}

private void exitProgram() {

if (mainClass != null) {

try {

// Вызываем метод main() в основном классе для завершения программы

Method mainMethod = mainClass.getMethod("main", String[].class);

mainMethod.invoke(null, (Object) new String[0]);

} catch (NoSuchMethodException | IllegalAccessException | InvocationTargetException e) {

e.printStackTrace();

}

}

}

}FileClient.java

package clientFileSend;

import com.sun.tools.javac.Main;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.io.\*;

import java.io.DataInputStream;

import java.lang.reflect.InvocationTargetException;

import java.lang.reflect.Method;

import java.net.Socket;

public class FileClient {

private String serverIp; // Адрес сервера

private int serverPort; // Порт сервера

private Class<?> mainClass; // Ссылка на класс Main

private static final String SAVE\_FOLDER = "client\_files"; // Папка для сохранения файлов на клиенте

public FileClient(String serverIp, int serverPort, Class<?> mainClass) {

this.serverIp = serverIp;

this.serverPort = serverPort;

this.mainClass = mainClass;

}

public void File() {

SwingUtilities.invokeLater(new Runnable() {

public void run() {

createAndShowGUI();

}

});

}

private void createAndShowGUI() {

JFrame frame = new JFrame("File Transfer Client");

frame.setDefaultCloseOperation(JFrame.DO\_NOTHING\_ON\_CLOSE);

frame.addWindowListener(new WindowAdapter() {

@Override

public void windowClosing(WindowEvent e) {

exitProgram();

}

});

JButton uploadButton = new JButton("Upload File");

uploadButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

JFileChooser fileChooser = new JFileChooser();

int result = fileChooser.showOpenDialog(frame);

if (result == JFileChooser.APPROVE\_OPTION) {

File selectedFile = fileChooser.getSelectedFile();

uploadFile(selectedFile);

}

}

});

JButton quitButton = new JButton("Quit");

quitButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

frame.dispose();

exitProgram();

}

});

JButton downloadButton = new JButton("Download File");

downloadButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

JFileChooser fileChooser = new JFileChooser();

int result = fileChooser.showSaveDialog(frame);

if (result == JFileChooser.APPROVE\_OPTION) {

File selectedFile = fileChooser.getSelectedFile();

downloadFile(selectedFile);

}

}

});

JButton viewButton = new JButton("View Files on Server");

viewButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

viewFilesOnServer();

}

});

frame.setLayout(new FlowLayout());

frame.add(uploadButton);

frame.add(downloadButton);

frame.add(viewButton);

frame.add(quitButton);

frame.pack();

frame.setVisible(true);

}

private void uploadFile(File file) {

try {

Socket socket = new Socket(serverIp, serverPort);

System.out.println("Соединение установлено: " + socket);

// Получаем выходной поток для отправки данных на сервер

OutputStream outputStream = socket.getOutputStream();

DataOutputStream dataOutputStream = new DataOutputStream(outputStream);

// Отправляем команду серверу (UPLOAD - загрузить файл)

dataOutputStream.writeUTF("UPLOAD");

// Отправляем имя файла на сервер

dataOutputStream.writeUTF(file.getName());

// Читаем данные из файла и отправляем на сервер

FileInputStream fileInputStream = new FileInputStream(file);

byte[] buffer = new byte[4096];

int bytesRead;

while ((bytesRead = fileInputStream.read(buffer)) != -1) {

dataOutputStream.write(buffer, 0, bytesRead);

}

// Закрываем потоки

fileInputStream.close();

dataOutputStream.close();

socket.close();

System.out.println("Файл успешно загружен на сервер: " + file.getAbsolutePath());

} catch (IOException e) {

e.printStackTrace();

}

}

private void downloadFile(File file) {

try {

Socket socket = new Socket(serverIp, serverPort);

System.out.println("Соединение установлено: " + socket);

// Получаем выходной поток для отправки данных на сервер

OutputStream outputStream = socket.getOutputStream();

DataOutputStream dataOutputStream = new DataOutputStream(outputStream);

// Отправляем команду серверу (DOWNLOAD - скачать файл)

dataOutputStream.writeUTF("DOWNLOAD");

// Отправляем имя файла на сервер

dataOutputStream.writeUTF(file.getName());

// Получаем входной поток для чтения данных с сервера

InputStream inputStream = socket.getInputStream();

DataInputStream dataInputStream = new DataInputStream(inputStream);

// Получаем размер файла

long fileSize = dataInputStream.readLong();

// Если размер файла равен -1, значит файл не существует на сервере

if (fileSize == -1) {

System.out.println("Файл не существует на сервере: " + file.getName());

return;

}

// Выбираем папку для сохранения файла на клиенте

JFileChooser fileChooser = new JFileChooser();

fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES\_ONLY);

int result = fileChooser.showSaveDialog(null);

if (result == JFileChooser.APPROVE\_OPTION) {

File saveFolder = fileChooser.getSelectedFile();

// Создаем файл для сохранения данных с сервера

File outputFile = new File(saveFolder, file.getName());

FileOutputStream fileOutputStream = new FileOutputStream(outputFile);

byte[] buffer = new byte[4096];

int bytesRead;

long totalBytesRead = 0;

while (totalBytesRead < fileSize && (bytesRead = dataInputStream.read(buffer, 0, (int) Math.min(buffer.length, fileSize - totalBytesRead))) != -1) {

fileOutputStream.write(buffer, 0, bytesRead);

totalBytesRead += bytesRead;

}

// Закрываем потоки

fileOutputStream.close();

dataInputStream.close();

dataOutputStream.close();

socket.close();

System.out.println("Файл успешно скачан с сервера и сохранен: " + outputFile.getAbsolutePath());

} else {

System.out.println("Загрузка файла отменена");

}

} catch (IOException e) {

e.printStackTrace();

}

}

private void viewFilesOnServer() {

try {

Socket socket = new Socket(serverIp, serverPort);

System.out.println("Соединение установлено: " + socket);

// Получаем выходной поток для отправки данных на сервер

OutputStream outputStream = socket.getOutputStream();

DataOutputStream dataOutputStream = new DataOutputStream(outputStream);

// Отправляем команду серверу (VIEW - просмотреть файлы)

dataOutputStream.writeUTF("VIEW");

// Получаем входной поток для чтения данных с сервера

InputStream inputStream = socket.getInputStream();

DataInputStream dataInputStream = new DataInputStream(inputStream);

// Получаем количество файлов на сервере

int fileCount = dataInputStream.readInt();

System.out.println("Список файлов на сервере:");

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

String fileName = dataInputStream.readUTF();

System.out.println(fileName);

}

dataInputStream.close();

dataOutputStream.close();

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

private void exitProgram() {

try {

Socket socket = new Socket(serverIp, serverPort);

System.out.println("Соединение установлено: " + socket);

// Получаем выходной поток для отправки данных на сервер

OutputStream outputStream = socket.getOutputStream();

DataOutputStream dataOutputStream = new DataOutputStream(outputStream);

// Отправляем команду серверу (EXIT - завершить программу)

dataOutputStream.writeUTF("EXIT");

// Закрываем соединение с сервером и ресурсы

dataOutputStream.close();

socket.close();

//так эта хрень не выходит

//System.exit(0);

Method mainMethod = mainClass.getMethod("main", String[].class);

mainMethod.invoke(null, new Object[] { new String[0] });

} catch (NoSuchMethodException | IllegalAccessException | InvocationTargetException | IOException e) {

e.printStackTrace();

}

}

}

Main.java

import java.util.Scanner;

import clientChat.ChatClient;

import clientFileSend.FileClient;

public class Main {

private static String serverIp;

private static volatile boolean isRunning = true;

private static int serverPort;

private String command;

static Scanner scanner = new Scanner(System.in);

private static String Menu = "Приветствую клиент,при помощи команд вы сможете" +

"(только для начала пожалуйста введите какой порт использовать серверу и к какому Ip-адрессу подключаться) " + "\n"

+ "File - отправить файл на сервер" + "\n" + "Chat - начать диалог с сервером" + "\n"

+ "Quit - завершить программу";

public static void main(String[] args) {

try {

System.out.println("Введите IP-адресс сервера : ");

serverIp = scanner.next();

System.out.println("Введите порт сервера : ");

serverPort = scanner.nextInt();

while (isRunning) {

System.out.println("Введите что вы хотите сделать : ");

String command = scanner.next();

switch (command) {

case "File":

System.out.println("Начинаем отправку файлов...");

FileClient client = new FileClient(serverIp, serverPort,Main.class);

client.File();

break;

case "Chat":

System.out.println("Начинаем диалог...");

ChatClient chat = new ChatClient(serverIp, serverPort);

chat.setMainClass(Main.class);

chat.chat();

break;

case "Quit":

System.out.println("Выполнение программы завершено");

isRunning = false;

break;

default:

System.out.println("Введена неверная команда");

}

}

} catch (Exception e) {

e.printStackTrace();

}

}

}