***Dokumentacja***

**Osoba pracująca nad aplikacją:**

**Barbara Wnuk nr.albumu**

***Serwer***

Serwer tworzy po cztery obiekty trzech klas (Kon, Lama, Alpaka) i przechowuje je w mapie. Obsługuje maksymalnie pięciu klientów jednocześnie. Każdy klient jest identyfikowany przez unikalne id. Serwer wysyła klientom informacje o statusie połączenia (OK lub REFUSED) oraz obsługuje żądania klientów dotyczące przesyłania kolekcji obiektów.

1. Inicjalizacja mapy `data` z czterema obiektami dla każdej z klas: `Kon`, `Lama`, `Alpaka`.

2. Tworzenie `ServerSocket` na porcie 12345 i oczekiwanie na połączenia od klientów.

3. Akceptowanie połączeń od klientów, sprawdzanie limitu klientów (maksymalnie pięciu) i tworzenie nowego wątku dla każdego klienta.

4. Obsługa żądań klientów przez klasę `ClientHandler`, która przetwarza zapytania dotyczące kolekcji obiektów (`get\_Kon`, `get\_Lama`, `get\_Alpaka`), serializuje odpowiednie obiekty i przesyła je do klienta.

***Klient***

Klient łączy się z serwerem, przesyła swoje id i otrzymuje status połączenia. Jeśli połączenie zostanie zaakceptowane, klient może wysyłać żądania dotyczące kolekcji obiektów (`get\_Kon`, `get\_Lama`, `get\_Alpaka`). Klient odbiera zserializowane obiekty, deserializuje je i wypisuje na konsoli.

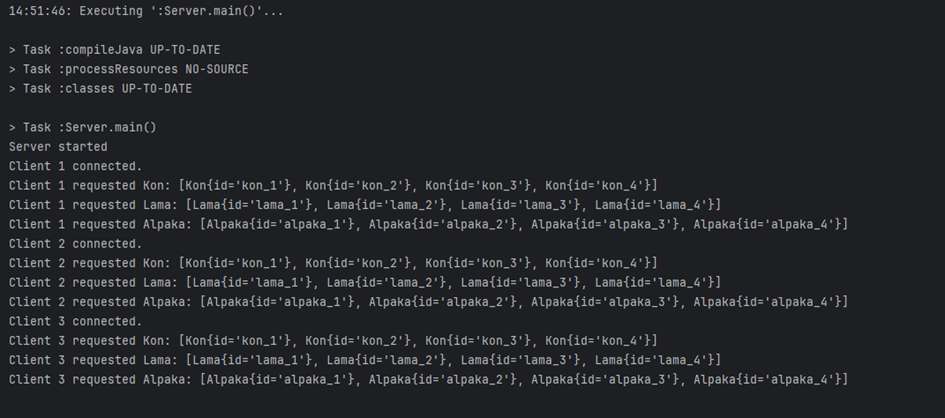
1. Połączenie z serwerem na porcie 12345 za pomocą `Socket`.

2. Odbieranie statusu połączenia (OK lub REFUSED) od serwera.

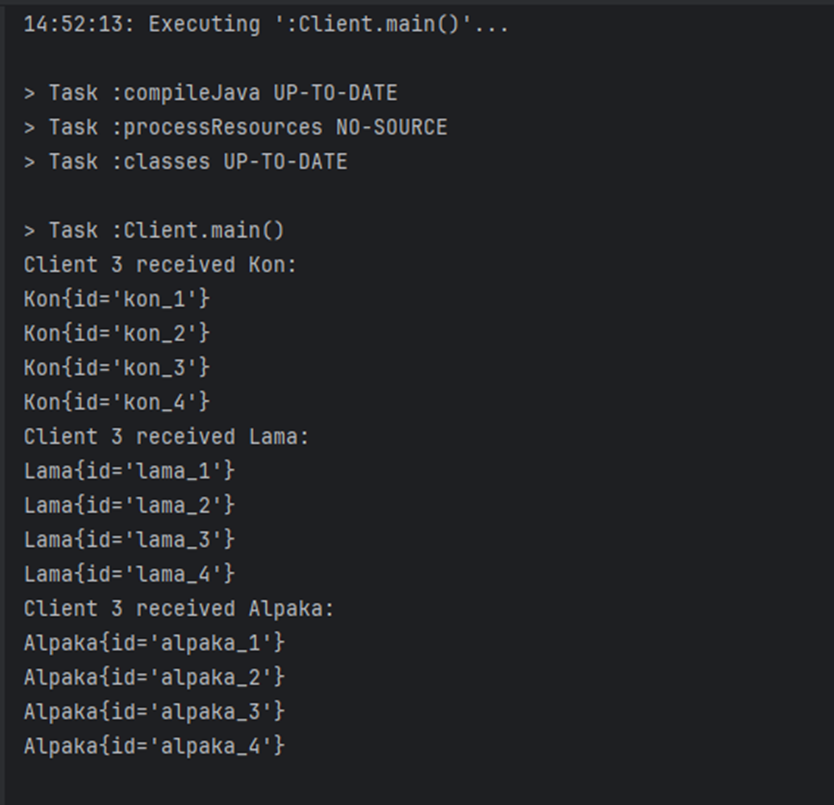
3. W przypadku akceptacji połączenia (OK), przesyłanie żądań dotyczących kolekcji obiektów.

4. Odbieranie zserializowanych obiektów od serwera, deserializacja i wypisywanie ich na konsoli.

**Client łączy się z server.main()**

****

**Start Client.main()**

****

***KOD APLIKACJI***

***Build.gradle.kts***

plugins {

id("java")

}

group = "org.example"

version = "1.0-SNAPSHOT"

repositories {

mavenCentral()

}

dependencies {

testImplementation("org.junit.jupiter:junit-jupiter-api:5.7.0")

testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:5.7.0")

}

tasks.test {

useJUnitPlatform()

}

***Server.java***

package org.example;

import java.io.\*;

import java.net.\*;

import java.util.\*;

import java.util.concurrent.\*;

class Kon {

String id;

Kon(String id) {

this.id = id;

}

@Override

public String toString() {

return "Kon{" + "id='" + id + '\'' + '}';

}

}

class Lama {

String id;

Lama(String id) {

this.id = id;

}

@Override

public String toString() {

return "Lama{" + "id='" + id + '\'' + '}';

}

}

class Alpaka {

String id;

Alpaka(String id) {

this.id = id;

}

@Override

public String toString() {

return "Alpaka{" + "id='" + id + '\'' + '}';

}

}

public class Server {

private static final int MAX\_CLIENTS = 5;

private static final Map<String, List<Object>> data = new ConcurrentHashMap<>();

private static final List<ClientHandler> clients = new ArrayList<>();

private static final Map<Integer, String> history = new ConcurrentHashMap<>();

private static int clientCounter = 0;

public static void main(String[] args) throws IOException {

data.put("Kon", Arrays.asList(new Kon("kon\_1"), new Kon("kon\_2"), new Kon("kon\_3"), new Kon("kon\_4")));

data.put("Lama", Arrays.asList(new Lama("lama\_1"), new Lama("lama\_2"), new Lama("lama\_3"), new Lama("lama\_4")));

data.put("Alpaka", Arrays.asList(new Alpaka("alpaka\_1"), new Alpaka("alpaka\_2"), new Alpaka("alpaka\_3"), new Alpaka("alpaka\_4")));

ServerSocket serverSocket = new ServerSocket(12345);

System.out.println("Server started");

while (true) {

Socket clientSocket = serverSocket.accept();

if (clients.size() >= MAX\_CLIENTS) {

new PrintWriter(clientSocket.getOutputStream(), true).println("REFUSED");

clientSocket.close();

continue;

}

clientCounter++;

ClientHandler clientHandler = new ClientHandler(clientSocket, clientCounter);

clients.add(clientHandler);

new Thread(clientHandler).start();

}

}

static class ClientHandler implements Runnable {

private final Socket clientSocket;

private final int clientId;

private boolean running = true;

ClientHandler(Socket socket, int id) {

this.clientSocket = socket;

this.clientId = id;

}

@Override

public void run() {

try (BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true)) {

out.println("OK");

out.println(clientId);

System.out.println("Client " + clientId + " connected.");

history.put(clientId, "Connected");

while (running) {

String request = in.readLine();

if (request == null) {

break;

}

handleRequest(request, out);

}

} catch (IOException e) {

e.printStackTrace();

} finally {

clients.remove(this);

}

}

private void handleRequest(String request, PrintWriter out) {

try {

Thread.sleep(new Random().nextInt(1000));

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

String[] parts = request.split("\_");

if (parts.length != 2 || !parts[0].equals("get")) {

out.println("ERROR");

return;

}

String className = parts[1];

List<Object> objects = data.get(className);

if (objects == null) {

out.println("ERROR");

return;

}

out.println(serializeObjects(objects));

history.put(clientId, "Requested " + className);

System.out.println("Client " + clientId + " requested " + className + ": " + objects);

}

private String serializeObjects(List<?> objects) {

StringBuilder sb = new StringBuilder();

for (Object obj : objects) {

sb.append(obj.toString()).append(";");

}

return sb.toString();

}

}

}

***Client.java***

package org.example;

import java.io.\*;

import java.net.\*;

public class Client {

public static void main(String[] args) throws IOException {

Socket socket = new Socket("localhost", 12345);

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

String response = in.readLine();

if ("OK".equals(response)) {

int clientId = Integer.parseInt(in.readLine());

requestAndPrintObjects(out, in, "get\_Kon", clientId);

requestAndPrintObjects(out, in, "get\_Lama", clientId);

requestAndPrintObjects(out, in, "get\_Alpaka", clientId);

} else {

System.out.println("Connection refused");

}

socket.close();

}

private static void requestAndPrintObjects(PrintWriter out, BufferedReader in, String request, int clientId) throws IOException {

out.println(request);

String response = in.readLine();

if (response.equals("ERROR")) {

System.out.println("Client " + clientId + " received an error.");

return;

}

String[] objects = response.split(";");

System.out.println("Client " + clientId + " received " + request.split("\_")[1] + ":");

for (String obj : objects) {

System.out.println(obj);

}