**Dokumentacja**

**Osoby pracujące nad Aplikacją:**

**Dawid Klimczuk** Nr.albumu**:** 159810

**Małgorzata Brzeska** Nr.albumu**:** 159831

**Mateusz Biernacki** Nr.albumu**:** 159838

**Opis Technologii**

* **Język programowania:** Java
* **Budowanie projektu:** Gradle
* **Biblioteki:** JUnit (do testów)

**Opis Aplikacji**

**Serwer**

Serwer tworzy po cztery obiekty trzech klas (Kot, Pies, Krolik) 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.

**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\_Kot`, `get\_Pies`, `get\_Krolik`). Klient odbiera zserializowane obiekty, deserializuje je i wypisuje na konsoli.

## Zrzuty ekranu

## Start Server.main()

### 

### Start Client.main()

### 

### Łączenie klientów z Server.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 Kot {  
 String id;  
  
 Kot(String id) {  
 this.id = id;  
 }  
  
 @Override  
 public String toString() {  
 return "Kot{" + "id='" + id + '\'' + '}';  
 }  
}  
  
class Pies {  
 String id;  
  
 Pies(String id) {  
 this.id = id;  
 }  
  
 @Override  
 public String toString() {  
 return "Pies{" + "id='" + id + '\'' + '}';  
 }  
}  
  
class Krolik {  
 String id;  
  
 Krolik(String id) {  
 this.id = id;  
 }  
  
 @Override  
 public String toString() {  
 return "Krolik{" + "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("Kot", Arrays.*asList*(new Kot("kot\_1"), new Kot("kot\_2"), new Kot("kot\_3"), new Kot("kot\_4")));  
 *data*.put("Pies", Arrays.*asList*(new Pies("pies\_1"), new Pies("pies\_2"), new Pies("pies\_3"), new Pies("pies\_4")));  
 *data*.put("Krolik", Arrays.*asList*(new Krolik("krolik\_1"), new Krolik("krolik\_2"), new Krolik("krolik\_3"), new Krolik("krolik\_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.\*;  
import java.util.Random;  
  
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\_Kot", clientId);  
 *requestAndPrintObjects*(out, in, "get\_Pies", clientId);  
 *requestAndPrintObjects*(out, in, "get\_Krolik", 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);  
 }  
 }  
}