**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ “НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”**

Курсовая работа

по дисциплине «Технологии и методы программирования»

Моделирующая программа «Волчий остров».

Группа: **АВТ-043**

Студент: **Калиниченко А. Е.**

Преподаватель: **Дубков И. С.**

Новосибирск 2022

1. **Цель**

Реализовать моделирующую программу по заданию и добавить клиент-серверную часть и базу данных.

1. **Задание**

Волчий остров размером 20х20 заселен дикими кроликами, волками и волчицами. Имеется по нескольку представителей каждого вида (случайное число от заданного). Кролики довольно глупы: в каждый момент времени они с одинаковой вероятностью 1/9 передвигаются в один из восьми соседних квадратов (за исключением участков, ограниченных береговой линией) или просто сидят неподвижно. Каждый кролик с вероятностью 0,2 превращается в двух кроликов. Каждая волчица передвигается случайным образом, пока в одном из соседних восьми квадратов не окажется кролик, за которым она охотится. Если волчица и кролик оказываются в одном квадрате, волчица съедает кролика и получает одно очко. В противном случае она теряет 0,1 очка. Волки и волчицы с нулевым количеством очков умирают.

В начальный момент времени все волки и волчицы имеют 1 очко. Волк ведет себя подобно волчице до тех пор, пока в соседних квадратах не исчезнут все кролики; тогда, если волчица находится в одном из восьми близлежащих квадратов, волк гонится за ней. Если волк и волчица окажутся в одном квадрате и там нет кролика, которого нужно съесть, они производят потомство случайного пола.

1. **Проектирование программы**

* Клиентская часть.
* Main.java

Инициализация окна и карты.

* Animal.java

Служит родительским классом для классов FWolf, MWolf и Rabbit. Позволяет узнать текущий голод (для волков), съесть кролика (для волков), узнать умер ли объект от голода (для волков), узнать двигался ли объект, узнать тип объекта, вычислить позицию для перемещения.

* FWolf.java

Реализация поведение волчицы.

* MWolf.java

Реализация поведения волка.

* Rabbit.java

Реализация поведения кролика.

* DrawMap.java

Отрисовка позиций животных.

* Client.java

Реализация сетевой клиентской части. Позволяет отправлять статистику по запросу от сервера.

* Серверная часть.
* Main.java

Инициализация сервера и базы данных (создание таблицы, если её нет). Уходит в бесконечный цикл для отслеживания подключения клиентов.

* ReadFromConsole.java

Происходит запуск дочернего потока, который считывает команды пользователя с консоли. Так же производит действия, согласно введённым командам.

* ClientThread.java

Используется для отправки запроса отдельному клиенту и получения статистики от него.

* DataBase.java

Реализует взаимодействие с базой данных. Создание таблицы, обновление записи, добавление записи, удаление всех записей, получение записи.

1. **Пример работы программы**

Клиент:

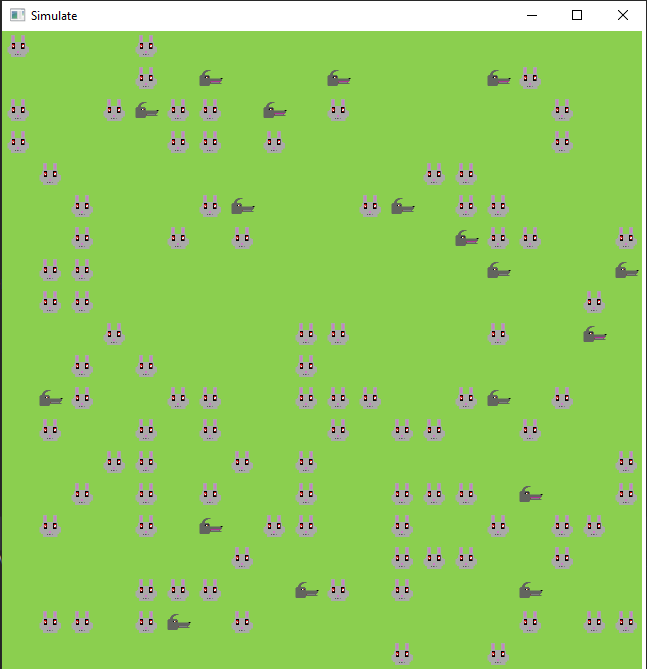


Рис. 1. Начало симуляции.

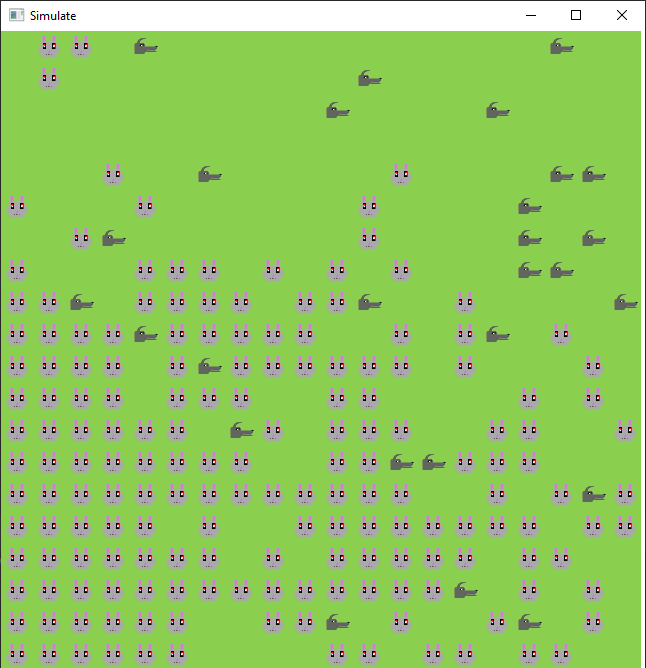


Рис. 2. Симуляция через 30 секунд.

Сервер:

Будет подключено 3 клиента.

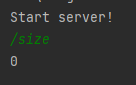


Рис. 3. Начало запуска сервера и просмотр количества подключенных клиентов.



Рис. 4. Просмотр количества подключенных клиентов.

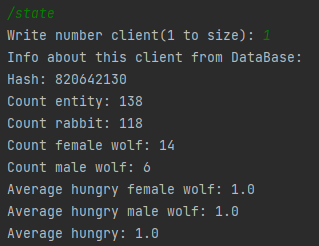


Рис. 5. Статистика первого клиента (на момент подключения к серверу).

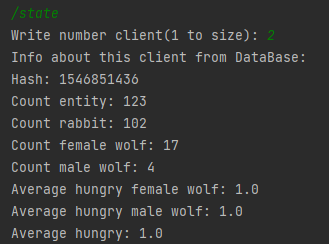


Рис. 6. Статистика второго клиента (на момент подключения к серверу).

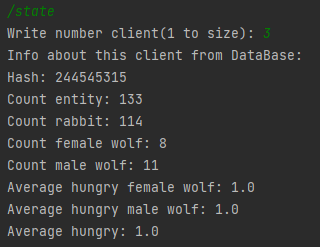


Рис. 7. Статистика третьего клиента (на момент подключения к серверу).

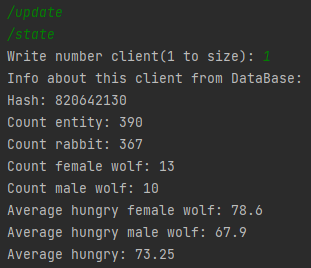


Рис. 8. Статистика первого клиента (после команды /update)

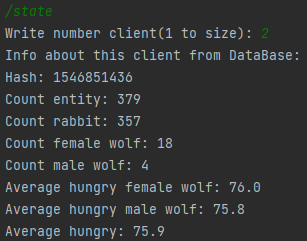


Рис. 9. Статистика второго клиента (после команды /update)

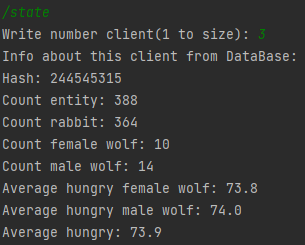


Рис. 10. Статистика третьего клиента (после команды /update)

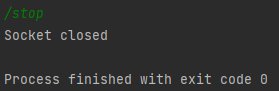


Рис. 11. Остановка сервера и очищение базы данных.

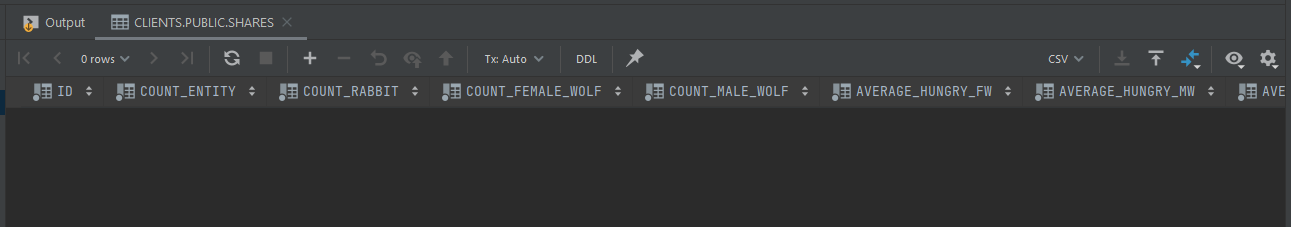


Рис. 12. Просмотр базы данных после остановки сервера.

1. **Вывод**

В ходе данной курсовой работы была выполнена реализация клиент-серверного приложения с использованием базы данных. На стороне клиента была реализована симуляция поведение волков и кроликов, а на стороне сервера сбор статистики клиентов и занесение её в базу данных, так же обновление этой базы данных.

Программа написана с помощью javafx, для реализации клиент-серверного соединения были использованы сокеты, для базы данных был использован h2.

**Приложение**

Клиент:

* **Main.java**

package main.java.Client;  
  
import javafx.application.Application;  
import javafx.scene.Group;  
import javafx.scene.Scene;  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
import javafx.stage.Stage;  
  
  
public class Main extends Application {  
  
 public Animal[][] map = new Animal[20][20];  
 public ImageView[][] imageMap = new ImageView[20][20];  
 public Client client;  
 DrawMap dm;  
  
 public void initMap()  
 {  
 for(int i = 0; i < 20; i++)  
 for(int j = 0; j < 20; j++)  
 {  
 if(Math.*random*() >= 0.3)  
 map[i][j] = new Animal();  
 else if(Math.*random*() >= 0.2)  
 map[i][j] = new Rabbit();  
 else if(Math.*random*() >= 0.4)  
 map[i][j] = new FWolf();  
 else  
 map[i][j] = new MWolf();  
  
  
 imageMap[i][j] = new ImageView(new Image("/main/resources/place.png"));  
 }  
 }  
  
 @Override  
 public void start(final Stage pS)  
 {  
 initMap();  
 client = new Client(map);  
 Group root = new Group();  
  
 for(int i = 0; i < 20; i++)  
 for(int j = 0; j < 20; j++)  
 {  
 imageMap[i][j].setFitHeight(32);  
 imageMap[i][j].setFitWidth(32);  
 imageMap[i][j].setLayoutX(i \* 32);  
 imageMap[i][j].setLayoutY(j \* 32);  
 root.getChildren().add(imageMap[i][j]);  
 }  
  
 Scene scene = new Scene(root);  
 pS.setScene(scene);  
  
 dm = new DrawMap(map, imageMap);  
  
 pS.setHeight(679);  
 pS.setWidth(660);  
 pS.setTitle("Simulate");  
 pS.show();  
 }  
  
 public static void main(String[] args) {  
 *launch*();  
 }  
}

* **Animal.java**

package main.java.Client;  
  
public class Animal  
{  
 Animal()  
 {  
 this.haveMove = false;  
 }  
  
 public void eat() {  
  
 }  
 public float currentHungry(){return -1;}  
 public boolean isDeath()  
 {  
 return false;  
 }  
  
 public void noMove()  
 {  
 this.haveMove = false;  
 }  
  
 public void yesMove()  
 {  
 this.haveMove = true;  
 }  
  
 public boolean isMove()  
 {  
 return this.haveMove;  
 }  
  
 public char getType()  
 {  
 return 'A';  
 }  
  
 public int[] move(int i, int j, Animal[][] map)  
 {  
 int[] tmp = new int[2];  
 tmp[0] = i;  
 tmp[1] = j;  
 return tmp;  
 }  
  
 protected boolean haveMove;  
}

* **FWolf.java**

package main.java.Client;  
  
import java.util.ArrayList;  
  
public class FWolf extends Animal  
{  
 FWolf()  
 {  
 super();  
 this.hungry = 1;  
 }  
  
 @Override  
 public char getType()  
 {  
 return 'F';  
 }  
  
 @Override  
 public boolean isDeath()  
 {  
 this.hungry -= 0.1;  
 return this.hungry <= 0;  
 }  
  
 @Override  
 public void eat()  
 {  
 this.hungry += 1;  
 }  
  
 @Override  
 public float currentHungry()  
 {  
 return this.hungry;  
 }  
  
 @Override  
 public int[] move(int i, int j, Animal[][] map)  
 {  
 int[] tmp = new int[2];  
  
 ArrayList<int[]> posRabbit = new ArrayList<>();  
 ArrayList<int[]> pos = new ArrayList<>();  
  
 for(int i1 = i-1; i1 < i+2; i1++)  
 for(int j1 = j-1; j1 < j+2; j1++)  
 if(i1 > -1 && i1 < 20 && j1 > -1 && j1 < 20 && (j1 != j || i1 != i))  
 if(map[i1][j1].getType() == 'A')  
 pos.add(new int[]{i1, j1});  
 else if(map[i1][j1].getType() == 'R')  
 posRabbit.add(new int[]{i1, j1});  
  
 if(!posRabbit.isEmpty())  
 {  
 int o = (int)(Math.*random*() \* posRabbit.size());  
 tmp[0] = posRabbit.get(o)[0];  
 tmp[1] = posRabbit.get(o)[1];  
 }  
 else if(!pos.isEmpty())  
 {  
 int o = (int)(Math.*random*() \* pos.size());  
 tmp[0] = pos.get(o)[0];  
 tmp[1] = pos.get(o)[1];  
 }  
 else  
 {  
 tmp[0] = i;  
 tmp[1] = j;  
 }  
  
  
 return tmp;  
 }  
  
 private float hungry;  
}

* **MWolf.java**

package main.java.Client;  
  
import java.util.ArrayList;  
  
public class MWolf extends Animal  
{  
 MWolf()  
 {  
 super();  
 this.hungry = 1;  
 }  
  
 @Override  
 public char getType()  
 {  
 return 'M';  
 }  
  
 @Override  
 public void eat()  
 {  
 this.hungry += 1;  
 }  
  
 @Override  
 public float currentHungry()  
 {  
 return this.hungry;  
 }  
  
 @Override  
 public boolean isDeath()  
 {  
 this.hungry -= 0.1;  
 return this.hungry <= 0;  
 }  
  
 @Override  
 public int[] move(int i, int j, Animal[][] map)  
 {  
 int[] tmp = new int[2];  
  
 ArrayList<int[]> pos = new ArrayList<>();  
 ArrayList<int[]> posRabbit = new ArrayList<>();  
 ArrayList<int[]> posF = new ArrayList<>();  
  
 for(int i1 = i-1; i1 < i+2; i1++)  
 for(int j1 = j-1; j1 < j+2; j1++)  
 if(i1 > -1 && i1 < 20 && j1 > -1 && j1 < 20 && (j1 != j || i1 != i))  
 if(map[i1][j1].getType() == 'A')  
 pos.add(new int[]{i1, j1});  
 else if(map[i1][j1].getType() == 'R')  
 posRabbit.add(new int[]{i1, j1});  
 else if(map[i1][j1].getType() == 'F')  
 posF.add(new int[]{i1, j1});  
  
 if(!posRabbit.isEmpty())  
 {  
 int o = (int)(Math.*random*() \* posRabbit.size());  
 tmp[0] = posRabbit.get(o)[0];  
 tmp[1] = posRabbit.get(o)[1];  
 }  
 else if(!posF.isEmpty())  
 {  
 int o = (int)(Math.*random*() \* posF.size());  
 tmp[0] = posF.get(o)[0];  
 tmp[1] = posF.get(o)[1];  
 }  
 else if(!pos.isEmpty())  
 {  
 int o = (int)(Math.*random*() \* pos.size());  
 tmp[0] = pos.get(o)[0];  
 tmp[1] = pos.get(o)[1];  
 }  
 else  
 {  
 tmp[0] = i;  
 tmp[1] = j;  
 }  
  
 return tmp;  
 }  
  
 private float hungry;  
}

* **Rabbit.java**

package main.java.Client;  
  
import java.util.ArrayList;  
  
public class Rabbit extends Animal  
{  
 Rabbit()  
 {  
 super();  
 }  
  
 @Override  
 public char getType()  
 {  
 return 'R';  
 }  
  
 @Override  
 public int[] move(int i, int j, Animal[][] map)  
 {  
 int[] tmp = new int[2];  
  
 ArrayList<int[]> pos = new ArrayList<>();  
  
 for(int i1 = i - 1; i1 < i + 2; i1++)  
 for(int j1 = j - 1; j1 < j + 2; j1++)  
 if(i1 > -1 && i1 < 20 && j1 > -1 && j1 < 20 && (j1 != j || i1 != i) && map[i1][j1].getType() == 'A')  
 pos.add(new int[]{i1, j1});  
  
  
 if(pos.size() == 0)  
 {  
 tmp[0] = i;  
 tmp[1] = j;  
 }  
 else  
 {  
 int o = (int)(Math.*random*() \* pos.size());  
 tmp[0] = pos.get(o)[0];  
 tmp[1] = pos.get(o)[1];  
 }  
  
  
 return tmp;  
 }  
}

* **DrawMap.java**

package main.java.Client;  
  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
  
public class DrawMap extends Thread  
{  
 DrawMap(Animal[][] map, ImageView[][] image\_Map)  
 {  
 this.map = map;  
 this.image\_Map = image\_Map;  
 this.updateMap();  
  
 start();  
 }  
  
 private void allowMove()  
 {  
 for(int i = 0; i <size\_map; i++)  
 for(int j = 0; j < size\_map; j++)  
 map[i][j].noMove();  
 }  
  
 private void updateMap()  
 {  
 for (int i = 0; i <size\_map; i++)  
 for (int j = 0; j < size\_map; j++)  
 {  
 switch (map[i][j].getType()) {  
 case 'A' -> image\_Map[i][j].setImage(new Image("/main/resources/place.png"));  
 case 'R' -> image\_Map[i][j].setImage(new Image("/main/resources/rabbit.png"));  
 case 'F' -> image\_Map[i][j].setImage(new Image("/main/resources/wolfF.png"));  
 case 'M' -> image\_Map[i][j].setImage(new Image("/main/resources/wolfM.png"));  
 }  
 }  
 }  
  
 private void updateRabbit()  
 {  
 synchronized (this.map)  
 {  
 for (int i = 0; i < size\_map; i++)  
 for (int j = 0; j < size\_map; j++)  
 if (map[i][j].getType() == 'R' && !map[i][j].isMove()) {  
 int[] tmp = map[i][j].move(i, j, map);  
 map[i][j].yesMove();  
 if (tmp[0] != i || tmp[1] != j) {  
 map[tmp[0]][tmp[1]] = map[i][j];  
  
 if (Math.*random*() <= 0.2)  
 map[i][j] = new Rabbit();  
 else  
 map[i][j] = new Animal();  
 }  
  
 }  
 }  
 }  
  
 public void updateWolfF()  
 {  
 synchronized (this.map) {  
 for (int i = 0; i < size\_map; i++)  
 for (int j = 0; j < size\_map; j++)  
 if (map[i][j].getType() == 'F' && !map[i][j].isMove()) {  
 if (!map[i][j].isDeath()) {  
 int[] tmp = map[i][j].move(i, j, map);  
 map[i][j].yesMove();  
  
 if (tmp[0] != i || tmp[1] != j) {  
 if (this.map[tmp[0]][tmp[1]].getType() == 'R')  
 this.map[i][j].eat();  
  
  
 map[tmp[0]][tmp[1]] = map[i][j];  
 map[i][j] = new Animal();  
 }  
 } else  
 map[i][j] = new Animal();  
  
 }  
 }  
 }  
  
 private void updateWolfM()  
 {  
 synchronized (this.map) {  
 for (int i = 0; i < this.size\_map; i++)  
 for (int j = 0; j < this.size\_map; j++)  
 if (this.map[i][j].getType() == 'M' && !this.map[i][j].isMove()) {  
 if (!this.map[i][j].isDeath()) {  
 int[] tmp = this.map[i][j].move(i, j, this.map);  
 this.map[i][j].yesMove();  
  
 if (tmp[0] != i || tmp[1] != j) {  
 if (this.map[tmp[0]][tmp[1]].getType() != 'F') {  
 if (this.map[tmp[0]][tmp[1]].getType() == 'R')  
 this.map[i][j].eat();  
  
 this.map[tmp[0]][tmp[1]] = this.map[i][j];  
 this.map[i][j] = new Animal();  
 } else {  
 if (i == 0) {  
 if (Math.*random*() > 0.5)  
 this.map[i + 1][j] = new FWolf();  
  
 else  
 this.map[i + 1][j] = new MWolf();  
 } else {  
 if (Math.*random*() > 0.5)  
 this.map[i - 1][j] = new FWolf();  
 else  
 this.map[i - 1][j] = new MWolf();  
  
 }  
 }  
 }  
 } else  
 this.map[i][j] = new Animal();  
  
 }  
 }  
 }  
  
 @Override  
 public void run()  
 {  
 while(true)  
 {  
 updateRabbit();  
 updateMap();  
 try {  
 *sleep*(800);  
 } catch (InterruptedException e) {  
 System.*out*.println(e.getMessage());  
 }  
  
 updateWolfF();  
 updateMap();  
 try {  
 *sleep*(800);  
 } catch (InterruptedException e) {  
 System.*out*.println(e.getMessage());  
 }  
  
 updateWolfM();  
 updateMap();  
 try {  
 *sleep*(800);  
 } catch (InterruptedException e) {  
 System.*out*.println(e.getMessage());  
 }  
  
 allowMove();  
  
 }  
  
 }  
  
 private final int size\_map = 20;  
 private final Animal[][] map;  
 private final ImageView[][] image\_Map;  
  
}

* **Client.java**

package main.java.Client;  
  
import java.io.BufferedReader;  
import java.io.BufferedWriter;  
import java.io.InputStreamReader;  
import java.io.OutputStreamWriter;  
import java.net.Socket;  
  
public class Client extends Thread  
{  
 Client(Animal[][] map)  
 {  
 this.map = map;  
  
 try  
 {  
 client = new Socket("localhost", 4004);  
 in = new BufferedReader(new InputStreamReader(client.getInputStream()));  
 out = new BufferedWriter(new OutputStreamWriter(client.getOutputStream()));  
 }  
 catch (Exception e)  
 {  
 System.*out*.println(e.getMessage());  
 }  
 System.*out*.println("Client start!");  
 start();  
 }  
  
 @Override  
 public void run()  
 {  
 while(true)  
 {  
 try  
 {  
 String word = this.in.readLine();  
 System.*out*.println(word);  
  
 switch (word)  
 {  
 case "getState" -> reQuestState("STATE");  
 case "getStateUpp" -> reQuestState("STATEUPP");  
 case "close" -> System.*exit*(0);  
 }  
  
 }  
 catch(Exception e){  
 if(e.getMessage().equals("Connection reset") || e.getMessage().equals("Cannot invoke \"java.io.BufferedReader.readLine()\" because \"this.in\" is null"))  
 {  
 System.*out*.println("Server is close");  
 break;  
 }  
 else System.*out*.println(e.getMessage());  
 }  
 }  
 }  
  
 private void reQuestState(String state)  
 {  
 synchronized (this.map)  
 {  
 int c\_wf = 0, c\_wm = 0, c\_r = 0;  
 float h\_wf = 0, h\_wm = 0;  
  
 for(int i = 0; i < 20; i++)  
 for(int j = 0; j < 20; j++)  
 switch (this.map[i][j].getType())  
 {  
 case 'F' -> {  
 c\_wf++;  
 h\_wf += this.map[i][j].currentHungry();  
 }  
  
 case 'M' -> {  
 c\_wm++;  
 h\_wm += this.map[i][j].currentHungry();  
 }  
  
 case 'R' -> c\_r++;  
 }  
  
 if(c\_wf != 0)  
 h\_wf /= c\_wf;  
 else h\_wf = -1;  
  
 if(c\_wm != 0)  
 h\_wm /= c\_wm;  
 else h\_wm = -1;  
  
 writeToServer(state + "r"+ c\_r + "f" + c\_wf + "m" + c\_wm +"F" + String.*format*("%.1f", h\_wf) + "M" + String.*format*("%.1f",h\_wm));  
 }  
 }  
  
 private void writeToServer(String word)  
 {  
 try  
 {  
 this.out.write(word + "\n");  
 this.out.flush();  
 }  
 catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
  
 private final Animal[][] map;  
 private Socket client;  
 private BufferedReader in;  
 private BufferedWriter out;  
}

Сервер:

* **Main.java**

package main.java.Server;  
  
  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.ArrayList;  
  
  
public class Main  
{  
 public static ServerSocket *serv*;  
 static ReadFromConsole *rfc* = new ReadFromConsole();  
 public static final ArrayList<ClientThread> *clientList* = new ArrayList<>();  
 public static final String *DB\_URL* = "jdbc:h2:C:/Users/Rarka/Desktop/РГР + КУРСАЧИ/ProgKurse/db/clients";  
 public static DataBase *db*;  
  
 public static void main(String[] args)  
 {  
  
 try {  
 try {  
 *db* = new DataBase(*DB\_URL*);  
 *serv* = new ServerSocket(4004);  
 *db*.createTable();  
 System.*out*.println("Start server!");  
  
 while(true)  
 {  
 Socket client = *serv*.accept();  
 *clientList*.add(new ClientThread(client));  
 }  
  
 } finally {  
 *serv*.close();  
 *db*.close();  
 }  
 }  
 catch (Exception e)  
 {  
 System.*out*.println(e.getMessage());  
 }  
 }  
}

* **ClientThread.java**

package main.java.Server;  
  
import java.io.\*;  
import java.net.Socket;  
import static main.java.Server.Main.*clientList*;  
import static main.java.Server.Main.*db*;  
  
public class ClientThread extends Thread  
{  
 public Socket client;  
 private BufferedReader in;  
 private BufferedWriter out;  
  
 ClientThread(Socket client)  
 {  
 try  
 {  
 this.client = client;  
 in = new BufferedReader(new InputStreamReader(client.getInputStream()));  
 out = new BufferedWriter(new OutputStreamWriter(client.getOutputStream()));  
 }  
 catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 this.writeToClient("getState");  
 start();  
 }  
  
 @Override  
 public void run()  
 {  
 while(true)  
 {  
 try  
 {  
 String word = in.readLine();  
  
 if(word.contains("STATE"))  
 {  
 int c\_r = 0, c\_wf = 0, c\_wm = 0, c = 0;  
 float h\_f = -1, h\_m = -1, ah = -1;  
  
 c\_r = Integer.*parseInt*(word.substring(word.indexOf('r') + 1, word.indexOf('f')));  
 c\_wf = Integer.*parseInt*(word.substring(word.indexOf('f') + 1, word.indexOf('m')));  
 c\_wm = Integer.*parseInt*(word.substring(word.indexOf('m') + 1, word.indexOf('F')));  
  
 c = c\_r + c\_wf + c\_wm;  
  
 h\_f = Float.*parseFloat*(word.substring(word.indexOf('F') + 1, word.indexOf('M')).replace(',','.'));  
 h\_m = Float.*parseFloat*(word.substring(word.indexOf('M') + 1).replace(',','.'));  
  
 if(h\_f != -1 && h\_m != -1)  
 ah = (h\_f + h\_m) / 2;  
 else ah = Math.*max*(h\_f, h\_m);  
  
 if(!word.contains("STATEUPP"))  
 *db*.insert(this.hashCode() + ", " + c + ", " + c\_r + ", " + c\_wf + ", " +  
 c\_wm + ", " + h\_f + ", " + h\_m + ", " + ah);  
 else *db*.update(this.hashCode(), c, c\_r, c\_wf, c\_wm, h\_f, h\_m, ah);  
 }  
 }  
 catch (Exception e) {  
 if(e.getMessage().equals("Connection reset"))  
 {  
 *clientList*.remove(this);  
 break;  
 }  
 else System.*out*.println(e.getMessage());  
 }  
  
 }  
 }  
 public void writeToClient(String word)  
 {  
 try  
 {  
 this.out.write(word + "\n");  
 this.out.flush();  
 }  
 catch (Exception e){  
 System.*out*.println(e.getMessage());  
 }  
 }  
}

* **ReadFromConsole.java**

package main.java.Server;  
  
import java.util.Scanner;  
  
import static main.java.Server.Main.\*;  
  
public class ReadFromConsole extends Thread  
{  
 Scanner console = new Scanner(System.*in*);  
 ReadFromConsole()  
 {  
 start();  
 }  
  
 private void checkCommand(String command)  
 {  
 try {  
 switch (command) {  
 case "/size" -> System.*out*.println(*clientList*.size());  
 case "/state" -> getStateClient();  
 case "/update" -> updateAllDate();  
 case "/writeTo" -> writeTo();  
 case "/clear" -> *db*.clear();

case "/help" -> writeHelp();  
 case "/closeAllClients" -> closeClients();  
 case "/stop" -> stopProgram();  
 }  
 }catch (Exception e)  
 {  
 System.*out*.println(e.getMessage());  
 }  
 }

private void writeHelp()  
{  
 System.*out*.println("/size - current count clients\n" +  
 "/state - write state of client\n" +  
 "/update - update data base of clients\n" +  
 "/writeTo - write massage to client\n" +  
 "/clear - clear data base\n" +  
 "/closeAllClients - close all modulate of clients\n" +  
 "/stop - stop server and clear data base");  
}

private void closeClients() throws Exception  
 {  
 synchronized (*clientList*)  
 {  
 for(ClientThread cl: *clientList*)  
 cl.writeToClient("close");  
  
 *db*.clear();  
 }  
 }  
 private void stopProgram() throws Exception  
 {  
 *serv*.close();  
 *db*.clear();  
 *db*.close();  
 System.*exit*(0);  
 }  
 private void updateAllDate()  
 {  
 synchronized (*clientList*)  
 {  
 for(ClientThread cl: *clientList*)  
 cl.writeToClient("getStateUpp");  
 }  
 }  
  
 private void getStateClient() {  
 System.*out*.print("Write number client(1 to size): ");  
 int count = console.nextInt();  
  
 if (count > *clientList*.size() || count < 1)  
 System.*out*.println("Error! Number isn't right");  
 else {  
 try {  
 System.*out*.println("Info about this client from DataBase: ");  
 *db*.getInfo(*clientList*.get(count - 1).hashCode());  
 }catch (Exception e)  
 {  
 System.*out*.println(e.getMessage());  
 }  
 }  
 }  
  
 private void writeTo()  
 {  
 System.*out*.print("Write number client(1 to size): ");  
 int count = console.nextInt();

console.nextLine();  
  
 if(count > *clientList*.size() || count < 1)  
 System.*out*.println("Error! Number isn't right");  
 else  
 {  
 System.*out*.print("Your string for this client: ");  
 String word = console.nextLine();  
 *clientList*.get(count - 1).writeToClient(word);  
 System.*out*.println("Good!");  
 }  
 }  
  
 @Override  
 public void run()  
 {  
 while(true)  
 {  
 String command = console.nextLine();  
 checkCommand(command);  
 if(command.equals("/stop"))  
 break;  
 }  
 }  
}

* **DataBase.java**

package main.java.Server;  
  
import java.sql.\*;  
  
public class DataBase  
{  
 DataBase(String DB\_URL)  
 {  
 this.DB\_URL = DB\_URL;  
 try  
 {  
 *connection* = DriverManager.*getConnection*(DB\_URL);  
 }catch(Exception e)  
 {  
 System.*out*.println(e.getMessage());  
 }  
 }  
  
 public void close() {  
 try {  
 if (*connection* != null && !*connection*.isClosed())  
 *connection*.close();  
 } catch (SQLException e) {  
 System.*out*.println("Error of close SQL table");  
 }  
 }  
  
 private void reopenConnection() throws SQLException {  
 if (*connection* == null || *connection*.isClosed()) {  
 *connection* = DriverManager.*getConnection*(DB\_URL);  
 }  
 }  
  
 private void executeSqlStatement(String sql, String description) throws SQLException  
 {  
 this.reopenConnection();  
 Statement statement = *connection*.createStatement();  
 statement.execute(sql);  
 statement.close();  
  
 if (description != null)  
 System.*out*.println(description);  
 }  
  
 private void executeSqlStatement(String sql) throws SQLException  
 {  
 executeSqlStatement(sql, null);  
 };  
  
 public void createTable() throws SQLException  
 {  
 executeSqlStatement("CREATE TABLE IF NOT EXISTS shares(" +  
 "id INTEGER NOT NULL PRIMARY KEY," +  
 "count\_entity INTEGER NOT NULL," +  
 "count\_rabbit INTEGER NOT NULL," +  
 "count\_female\_wolf INTEGER NOT NULL," +  
 "count\_male\_wolf INTEGER NOT NULL," +  
 "average\_hungry\_fw DECIMAL(10,2) NOT NULL," +  
 "average\_hungry\_mw DECIMAL(10,2) NOT NULL," +  
 "average\_hungry\_all DECIMAL(10,2) NOT NULL)");  
 }  
  
 public void update(int hash, int c, int c\_r, int c\_wf, int c\_wm, float h\_f, float h\_m, float ah) throws SQLException  
 {  
 this.reopenConnection();  
 Statement statement = *connection*.createStatement();  
  
 statement.executeUpdate("UPDATE SHARES SET " +  
 "count\_entity=" + c +  
 ", count\_rabbit=" + c\_r +  
 ", count\_female\_wolf=" + c\_wf +  
 ", count\_male\_wolf=" + c\_wm +  
 ", average\_hungry\_fw=" + h\_f +  
 ", average\_hungry\_mw=" + h\_m +  
 ", average\_hungry\_all=" + ah + " WHERE ID=" + hash +";");  
  
 statement.close();  
 }  
 public void getInfo(int hash) throws SQLException  
 {  
 this.reopenConnection();  
 Statement statement = *connection*.createStatement();  
 ResultSet rS = statement.executeQuery("SELECT \* FROM SHARES WHERE ID=" + hash + ";");  
  
 while(rS.next())  
 {  
 System.*out*.println("Hash: " + rS.getInt(1));  
 System.*out*.println("Count entity: " + rS.getInt(2));  
 System.*out*.println("Count rabbit: " + rS.getInt(3));  
 System.*out*.println("Count female wolf: " + rS.getInt(4));  
 System.*out*.println("Count male wolf: " + rS.getInt(5));  
 System.*out*.println("Average hungry female wolf: " + rS.getFloat(6));  
 System.*out*.println("Average hungry male wolf: " + rS.getFloat(7));  
 System.*out*.println("Average hungry: " + rS.getFloat(8));  
 }  
  
 statement.close();  
 }  
 public void insert(String insert) throws SQLException  
 {  
 this.executeSqlStatement("INSERT INTO SHARES VALUES(" + insert + ");");  
 }  
 public void clear() throws SQLException  
 {  
 this.executeSqlStatement("DELETE FROM SHARES;");  
 }  
  
 private static Connection *connection*;  
 private String DB\_URL;  
}