**Содержательная постановка задачи: “Нечеткое управление скоростью вентилятора в зависимости от температуры и влажности в помещении”**

В помещении установлен вентилятор, который должен автоматически регулировать скорость вращения в зависимости от двух параметров: **температуры воздуха** (измеряется в °C) и **влажности воздуха** (измеряется в %). **Цель системы — обеспечить комфортные условия, избегая перегрева или чрезмерной сухости.**

Опыт использования вентилятора позволяет сформулировать несколько эвристических правил, которые мы применяем в случае регулирования скорости работы вентилятора:

1. Если температура низкая и влажность высокая, то вентилятор выключен.
2. Если температура средняя и влажность средняя, то вентилятор работает на средней скорости.
3. Если температура высокая и влажность низкая, то вентилятор работает на максимальной скорости.
4. Если температура высокая и влажность средняя, то вентилятор работает на быстрой скорости.
5. Если температура средняя и влажность низкая, то вентилятор работает на средней скорости.

Эта информация будет использоваться при построении базы правил системы нечеткого вывода, которая позволяет реализовать данную модель нечеткого управления.

**Построение базы нечетких лингвистических правил**

Для формирования базы правил систем нечеткого вывода необходимо предварительно определить входные и выходные лингвистические переменные. Очевидно, в качестве входной лингвистических переменной следует использовать температуру и влажность воздуха в помещении, формально: β1 — "температура воздуха", β2 — " влажность воздуха ". В качестве выходной лингвистической переменной будем использовать скорость работы вентилятора: β3 — "скорость работы".

В этом случае система нечеткого вывода будет содержать 5 правил нечетких продукций следующего вида:

ПРАВИЛО\_1: ЕСЛИ "температура воздуха низкая" И "влажность воздуха высокая" ТО "вентилятор выключен"

ПРАВИЛО\_2: ЕСЛИ "температура воздуха средняя" И "влажность воздуха средняя " ТО "вентилятор работает на средней скорости"

ПРАВИЛО\_3: ЕСЛИ "температура воздуха высокая" И "влажность воздуха низкая" ТО "вентилятор работает на максимальной скорости"

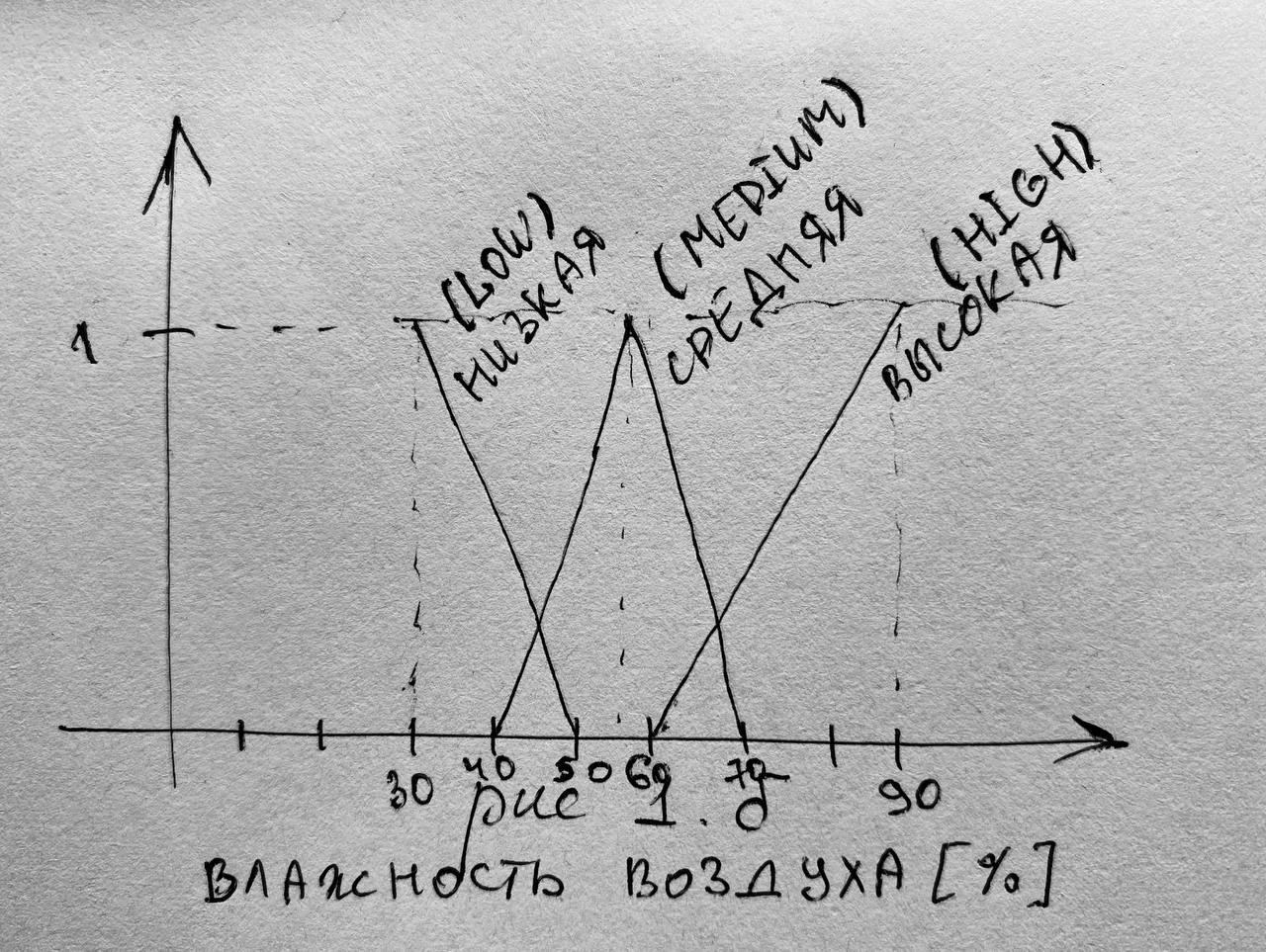
ПРАВИЛО\_4: ЕСЛИ "температура воздуха высокая" И "влажность воздуха средняя" ТО "вентилятор работает на быстрой скорости "

ПРАВИЛО\_5: ЕСЛИ "температура воздуха средняя" И "влажность воздуха низкая" ТО "вентилятор работает на средней скорости "

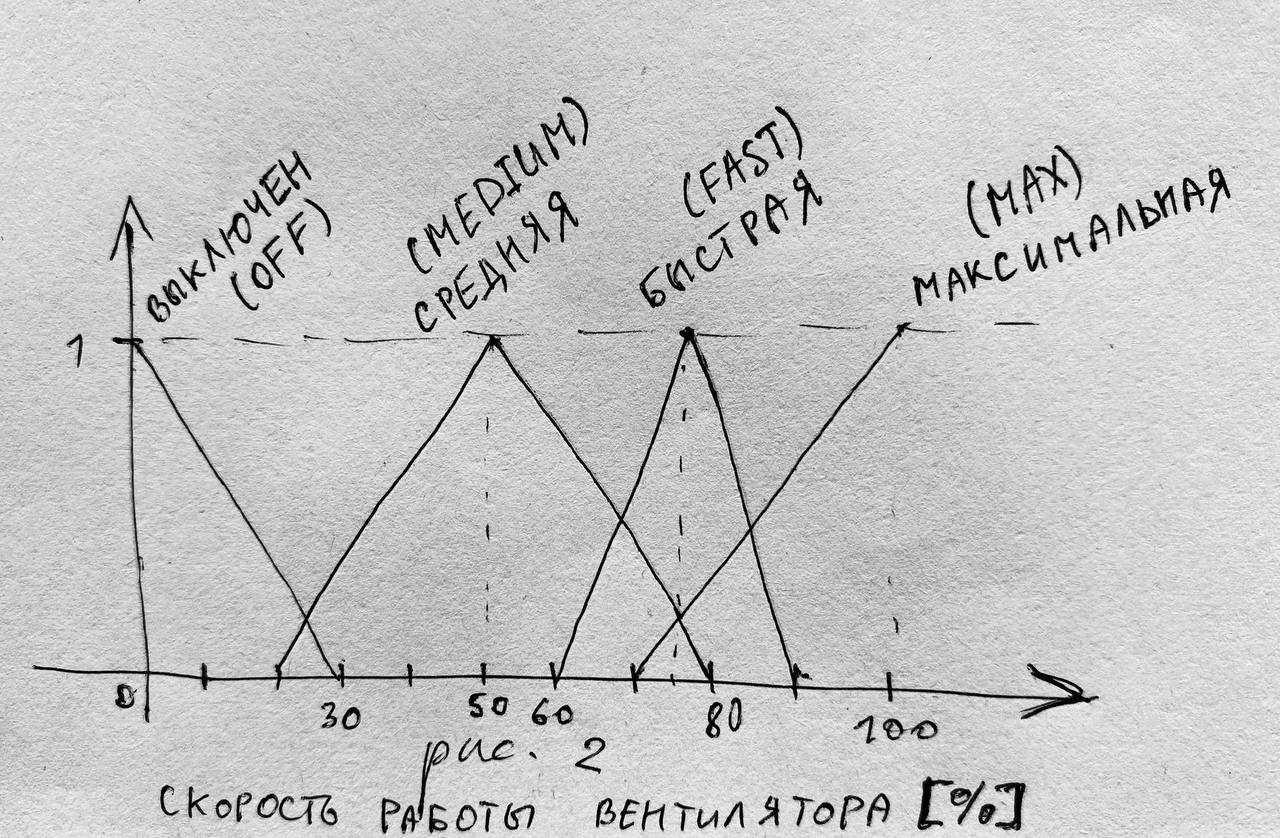
**Фаззификация входных переменных**

В качестве терм-множества первой лингвистической переменной будем использовать множество Т1 = {"низкая", "средняя", "высокая"} с функциями принадлежности, изображенными на рис. 1, а, в качестве терм-множества второй лингвистической переменной будем использовать множество Т2 = {"низкая", "средняя", "высокая"} с функциями принадлежности, изображенными на рис. 1, б.





В качестве терм-множества выходной лингвистической переменной будем использовать множество Т3 = {"выключен", "средняя", "быстрая", "максимальная"} с кусочно-линейными функциями принадлежности, изображенными на рис. 2.



При этом температура воздуха измеряется в градусах Цельсия, влажность воздуха в процентах, а скорость работы вентилятора — в процентах от максимально возможной скорости.

Используя в качестве алгоритма вывода алгоритм Мамдани, рассмотрим пример его выполнения для случая, когда текущая температура воздуха составляет **27°C**, а влажность - **45%**. В этом случае фаззификация входных лингвистических переменных приводит к следующим значениям степеней истинности:

Для температуры 27°C  
Низкая: μ(27) = 0  
Средняя: μ(27) = (30-27)/5 = 0.6  
Высокая: μ(27) = 0

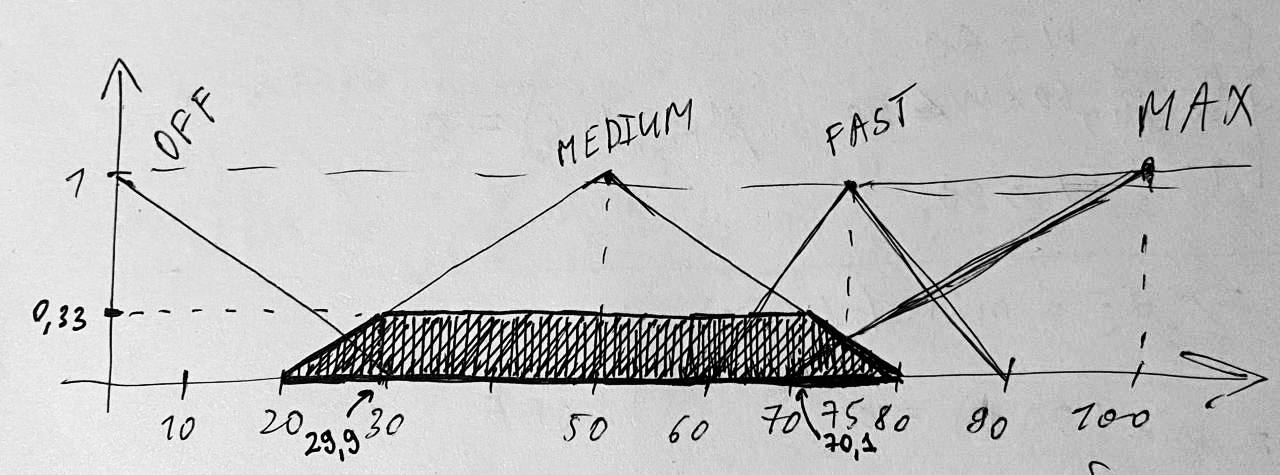
Для влажности 45%  
Низкая: μ(45) = (50-45)/20 = 0.25  
Средняя: μ(45) = min[(45-40)/15, (70-45)/15] = min[0.33, 1.67] = 0.33  
Высокая: μ(45) = 0

В результате анализа базы правил активными оказываются правила нечетких продукций с номерами 2 и 5, которые получают следующие степени активации:

Правило 2 (Средняя T и Средняя H): 0.33 => средняя скорость  
Правило 5 (Средняя T и Низкая H): 0.25 => средняя скорость

Поскольку все условия в правилах заданы в форме нечетких лингвистических высказываний первого вида, этап их агрегирования выполняется с использованием min-конъюнкции, что сохраняет вычисленные степени истинности без изменения. Эти активированные правила будут использоваться для дальнейшего процесса нечеткого вывода и определения оптимальной скорости работы вентилятора.

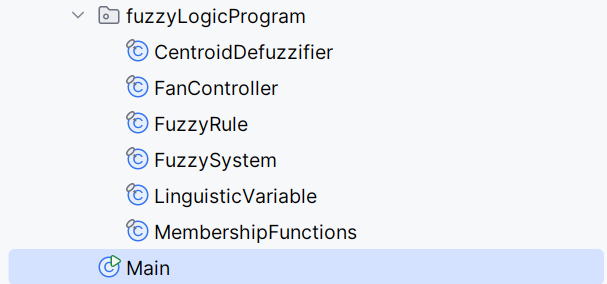
Следующим этапом нечеткого вывода является активизация заключений в нечетких правилах продукций. Функция принадлежности после аккумуляции:



Дефаззификация выходной лингвистической переменной "скорость работы" методом центра тяжести для значений функции принадлежности, приводит к значению управляющей переменной, равному средней скорости работы в 50%. Это значение и является результатом решения задачи нечеткого вывода для текущих значений входных лингвистических переменных "температура воздуха", " влажность воздуха ".

**А теперь реализуем решение задачи на языке программирования java:**

Получился следующий набор классов.



1. MembershipFunctions: Класс, содержащий стандартные функции принадлежности для нечеткой логики. Все функции возвращают степень принадлежности значения x к нечеткому множеству.

package fuzzyLogicProgram;  
public final class MembershipFunctions {  
 private MembershipFunctions() {}  
  
 */\*  
 Треугольная функция принадлежности  
 x - входное значение  
 a - левая граница (где μ=0)  
 b - вершина (где μ=1)  
 c - правая граница (где μ=0)  
 \*/* public static double triangular(double x, double a, double b, double c) {  
 if (x <= a || x >= c) return 0;  
 return x <= b ? (x - a) / (b - a) : (c - x) / (c - b);  
 }  
  
 */\*  
 Трапециевидная функция принадлежности  
 x - входное значение  
 a - начало подъема (где μ=0)  
 b - начало плато (где μ=1)  
 c - конец плато (где μ=1)  
 d - конец спада (где μ=0)  
 \*/* public static double trapezoidal(double x, double a, double b, double c, double d) {  
 if (x <= a || x >= d) return 0;  
 if (x >= b && x <= c) return 1;  
 return x < b ? (x - a) / (b - a) : (d - x) / (d - c);  
 }  
  
 */\*  
 Убывающая линейная функция принадлежности  
 x - входное значение  
 a - начало спада (где μ=1)  
 b - конец спада (где μ=0)  
 \*/* public static double decreasingLinear(double x, double a, double b) {  
 if (x <= a) return 1;  
 if (x >= b) return 0;  
 return (b - x) / (b - a);  
 }  
  
 */\*  
 Возрастающая линейная функция принадлежности  
 x - входное значение  
 a - начало подъема (где μ=0)  
 b - конец подъема (где μ=1)  
 \*/* public static double increasingLinear(double x, double a, double b) {  
 if (x <= a) return 0;  
 if (x >= b) return 1;  
 return (x - a) / (b - a);  
 }  
}

2. LinguisticVariable: Класс лингвистической переменной нечеткой логики. Содержит термы (нечеткие множества) с их функциями принадлежности.

package fuzzyLogicProgram;  
  
import java.util.Collections;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.Objects;  
import java.util.function.Function;  
  
public final class LinguisticVariable {  
 private final String name;  
 private final Map<String, Function<Double, Double>> terms;  
 *// Термы и их функции принадлежности* public LinguisticVariable(String name) { *//Конструктор* this.name = Objects.*requireNonNull*(name);  
 this.terms = new HashMap<>();  
 }  
  
 */\*  
 Добавление терма к переменной  
 termName - название терма  
 membershipFunction - функция принадлежности для терма  
 \*/* public void addTerm(String termName, Function<Double, Double> membershipFunction) {  
 terms.put(termName, membershipFunction);  
 }  
  
 *//Вычисление степени принадлежности значения к терму* public double getMembership(String termName, double value) {  
 Function<Double, Double> function = terms.get(termName);  
 if (function == null) {  
 throw new IllegalArgumentException("Unknown term: " + termName);  
 }  
 return function.apply(value);  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public Map<String, Function<Double, Double>> getTerms() {  
 return Collections.*unmodifiableMap*(terms);  
 }  
}

3. FuzzyRule: Класс нечеткого правила вида "ЕСЛИ условие ТО заключение"

package fuzzyLogicProgram;  
  
import java.util.List;  
import java.util.Map;

public final class FuzzyRule {  
 private final List<String> inputVariables;  
 private final List<String> inputTerms;  
 private final String outputTerm;  
  
 *//Конструктор правила для двух условий* public FuzzyRule(String inputVar1, String inputTerm1,  
 String inputVar2, String inputTerm2,  
 String outputTerm) {  
 this.inputVariables = List.*of*(inputVar1, inputVar2);  
 this.inputTerms = List.*of*(inputTerm1, inputTerm2);  
 this.outputTerm = outputTerm;  
 }  
  
 *//Вычисление степени активации правила* public double evaluate(Map<String, Map<String, Double>> inputs) {  
 double activation = 1.0;  
  
 for (int i = 0; i < inputVariables.size(); i++) {  
 String varName = inputVariables.get(i);  
 String termName = inputTerms.get(i);  
  
 Map<String, Double> varMemberships = inputs.get(varName);  
 if (varMemberships == null) {  
 return 0.0; *// Нет данных по терму* }  
  
 Double membership = varMemberships.get(termName);  
 if (membership == null) {  
 return 0.0; *// Нет данных по терму* }  
 *// Логическое И через минимум* activation = Math.*min*(activation, membership);  
 }  
  
 return activation;  
 }  
  
 public String getOutputTerm() {  
 return outputTerm;  
 }  
  
 public List<String> getInputVariables() {  
 return inputVariables;  
 }  
  
 public List<String> getInputTerms() {  
 return inputTerms;  
 }  
}

4. CentroidDefuzzifier: Класс для дефаззификации методом центра площади

package fuzzyLogicProgram;  
  
import java.util.Map;

public final class CentroidDefuzzifier {  
 private static final int *PRECISION* = 1000;*// Количество шагов интегрирования* private CentroidDefuzzifier() {}*// Запрещаем создание экземпляров  
  
 /\*  
 Дефаззификация методом центра площади  
 outputVar - выходная лингвистическая переменная  
 activatedTerms - активированные термы с их степенями  
 min - минимальное значение выходной переменной  
 max - максимальное значение выходной переменной  
 return: четкое значение выходной переменной  
 \*/* public static double defuzzify(LinguisticVariable outputVar,  
 Map<String, Double> activatedTerms,  
 double min, double max) {  
 return *defuzzifyWithLogging*(outputVar, activatedTerms, min, max, false);  
 }  
  
 *// Версия с логированием* public static double defuzzifyWithLogging(LinguisticVariable outputVar,  
 Map<String, Double> activatedTerms,  
 double min, double max) {  
 return *defuzzifyWithLogging*(outputVar, activatedTerms, min, max, true);  
 }  
  
 private static double defuzzifyWithLogging(LinguisticVariable outputVar,  
 Map<String, Double> activatedTerms,  
 double min, double max,  
 boolean logging) {  
 double step = (max - min) / *PRECISION*;  
 double numerator = 0.0;  
 double denominator = 0.0;  
  
 if (logging) {  
 System.*out*.println("\nПроцесс дефаззификации методом центра площади:");  
 System.*out*.println("x\tμ(x)\tx\*μ(x)");  
 System.*out*.println("---------------------");  
 }  
  
 for (double x = min; x <= max; x += step) {  
 double mu = *aggregatedMembership*(outputVar, activatedTerms, x);  
  
 if (logging && x % 10 == 0) { *// Выводим каждые 10 единиц для краткости* System.*out*.printf("%.1f\t%.3f\t%.3f\n", x, mu, x \* mu);  
 }  
  
 numerator += x \* mu \* step;  
 denominator += mu \* step;  
 }  
  
 if (logging) {  
 System.*out*.println("... (пропущены промежуточные значения)");  
 System.*out*.println("---------------------");  
 System.*out*.printf("Итого: ∫x\*μ(x)dx = %.3f, ∫μ(x)dx = %.3f\n", numerator, denominator);  
 System.*out*.printf("Центр площади = %.3f / %.3f = %.1f%%\n",  
 numerator, denominator, denominator == 0 ? 0 : numerator / denominator);  
 }  
  
 return denominator == 0 ? 0 : numerator / denominator;  
 }  
  
 */\*  
 Вычисление агрегированной функции принадлежности  
 return: максимальное значение из всех активированных термов (логическое ИЛИ через MAX)  
 \*/* private static double aggregatedMembership(LinguisticVariable outputVar,  
 Map<String, Double> activatedTerms,  
 double x) {  
 double maxMu = 0.0;  
 for (Map.Entry<String, Double> entry : activatedTerms.entrySet()) {  
 double termMu = outputVar.getMembership(entry.getKey(), x);  
 maxMu = Math.*max*(maxMu, Math.*min*(entry.getValue(), termMu));  
 *// MIN для активации, MAX для агрегации* }  
 return maxMu;  
 }  
}

5. FuzzySystem: Нечеткая система принятия решений (система нечеткого вывода)

package fuzzyLogicProgram;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
*/\*  
 Нечеткая система принятия решений (система нечеткого вывода)  
 \*/*public final class FuzzySystem {  
 private final List<LinguisticVariable> inputVariables = new ArrayList<>();  
 private final LinguisticVariable outputVariable;  
 private final List<FuzzyRule> rules = new ArrayList<>();  
 private final double outputMin;  
 private final double outputMax;  
  
 *//Конструктор* public FuzzySystem(LinguisticVariable outputVariable, double outputMin, double outputMax) {  
 this.outputVariable = outputVariable;  
 this.outputMin = outputMin;  
 this.outputMax = outputMax;  
 }  
  
 public void addInputVariable(LinguisticVariable variable) {  
  
 inputVariables.add(variable);  
 }  
  
 public void addRule(FuzzyRule rule) {  
 rules.add(rule);  
 }  
  
 */\*  
 Основной метод вычисления выхода системы  
 inputValues - четкие входные значения  
 return: четкое выходное значение  
 \*/* public double evaluate(Map<String, Double> inputValues) {  
 Map<String, Map<String, Double>> fuzzifiedInputs = fuzzifyInputs(inputValues);  
 Map<String, Double> outputActivations = activateRules(fuzzifiedInputs);  
 return CentroidDefuzzifier.*defuzzify*(outputVariable, outputActivations, outputMin, outputMax);  
 }  
  
 */\*  
 Фаззификация входных данных  
 return словарь: {переменная: {терм: степень принадлежности}}  
 \*/* public Map<String, Map<String, Double>> fuzzifyInputs(Map<String, Double> inputValues) {  
 Map<String, Map<String, Double>> result = new HashMap<>();  
  
 for (LinguisticVariable var : inputVariables) {  
 Map<String, Double> memberships = new HashMap<>();  
 String varName = var.getName();  
 double value = inputValues.get(varName);  
 *// Вычисляем принадлежности для всех термов переменной* var.getTerms().forEach((termName, function) -> {  
 memberships.put(termName, var.getMembership(termName, value));  
 });  
  
 result.put(varName, memberships);  
 }  
  
 return result;  
 }  
  
 *// Версии активации правил с логированием и без* public Map<String, Double> activateRules(Map<String, Map<String, Double>> fuzzifiedInputs) {  
 return activateRulesWithLogging(fuzzifiedInputs, false);  
 }  
  
 public Map<String, Double> activateRulesWithLogging(Map<String, Map<String, Double>> fuzzifiedInputs) {  
 return activateRulesWithLogging(fuzzifiedInputs, true);  
 }  
  
 */\*  
 Активация правил системы  
 return словарь: {выходной терм: степень активации}  
 \*/* private Map<String, Double> activateRulesWithLogging(Map<String, Map<String, Double>> fuzzifiedInputs,  
 boolean logging) {  
 Map<String, Double> outputActivations = new HashMap<>();  
  
 if (logging) {  
 System.*out*.println("\nАнализ базы правил:");  
 System.*out*.println("--------------------------------------------------");  
 System.*out*.println("№ | Условие | Активация");  
 System.*out*.println("--------------------------------------------------");  
 }  
  
 int ruleNum = 1;  
 for (FuzzyRule rule : rules) {  
 double activation = rule.evaluate(fuzzifiedInputs);  
  
 if (logging) {  
 String condition = String.*format*("Если %s=%s и %s=%s",  
 rule.getInputVariables().get(0), rule.getInputTerms().get(0),  
 rule.getInputVariables().get(1), rule.getInputTerms().get(1));  
  
 System.*out*.printf("%d | %-30s | %.2f\n",  
 ruleNum++, condition, activation);  
 }  
  
 if (activation > 0) {  
 String outputTerm = rule.getOutputTerm();  
 *// Агрегируем через MAX, если правило активировало тот же терм* outputActivations.merge(outputTerm, activation, Math::*max*);  
 }  
 }  
  
 if (logging && !outputActivations.isEmpty()) {  
 System.*out*.println("\nАктивированные выходные термины:");  
 outputActivations.forEach((term, activation) ->  
 System.*out*.printf("- %s: степень активации = %.2f\n", term, activation));  
 }  
  
 return outputActivations;  
 }  
}

6. FanController: Контроллер управления вентилятором на основе нечеткой логики

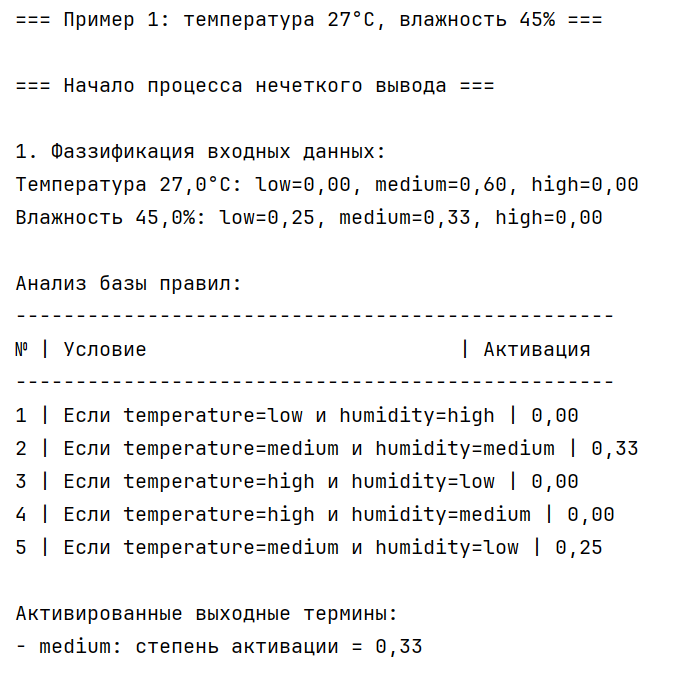
package fuzzyLogicProgram;  
  
import java.util.HashMap;  
import java.util.Map;  
  
*//Контроллер управления вентилятором на основе нечеткой логики*public final class FanController {  
 private final FuzzySystem fuzzySystem;  
 private final LinguisticVariable temperature;  
 private final LinguisticVariable humidity;  
 private final LinguisticVariable fanSpeed;  
  
 public FanController() {  
 *// Создаем лингвистические переменные* this.temperature = createTemperatureVariable();  
 this.humidity = createHumidityVariable();  
 this.fanSpeed = createFanSpeedVariable();  
  
 *// Инициализируем нечеткую систему* this.fuzzySystem = new FuzzySystem(fanSpeed, 0, 100);  
 this.fuzzySystem.addInputVariable(temperature);  
 this.fuzzySystem.addInputVariable(humidity);  
  
 *// Добавляем правила* createRules();  
 }  
  
 *// Создание лингвистической переменной для температуры* private LinguisticVariable createTemperatureVariable() {  
 LinguisticVariable temp = new LinguisticVariable("temperature");  
 *// Термы с их функциями принадлежности:* temp.addTerm("low", x -> MembershipFunctions.*decreasingLinear*(x, 15, 22));  
 temp.addTerm("medium", x -> MembershipFunctions.*triangular*(x, 20, 25, 30));  
 temp.addTerm("high", x -> MembershipFunctions.*increasingLinear*(x, 28, 35));  
 return temp;  
 }  
  
 *// Создание лингвистической переменной для влажности* private LinguisticVariable createHumidityVariable() {  
 LinguisticVariable hum = new LinguisticVariable("humidity");  
 hum.addTerm("low", x -> MembershipFunctions.*decreasingLinear*(x, 30, 50));  
 hum.addTerm("medium", x -> MembershipFunctions.*triangular*(x, 40, 55, 70));  
 hum.addTerm("high", x -> MembershipFunctions.*increasingLinear*(x, 60, 90));  
 return hum;  
 }  
  
 *// Создание лингвистической переменной для скорости вентилятора* private LinguisticVariable createFanSpeedVariable() {  
 LinguisticVariable speed = new LinguisticVariable("fan\_speed");  
 speed.addTerm("off", x -> MembershipFunctions.*decreasingLinear*(x, 0, 30));  
 speed.addTerm("medium", x -> MembershipFunctions.*triangular*(x, 20, 50, 80));  
 speed.addTerm("fast", x -> MembershipFunctions.*triangular*(x, 60, 75, 90));  
 speed.addTerm("max", x -> MembershipFunctions.*increasingLinear*(x, 70, 100));  
 return speed;  
 }  
  
 *// Создание базы правил нечеткой системы* private void createRules() {  
 *// Правило 1: Если температура низкая и влажность высокая → выключен* fuzzySystem.addRule(new FuzzyRule("temperature", "low", "humidity", "high", "off"));  
  
 *// Правило 2: Если температура средняя и влажность средняя → средняя скорость* fuzzySystem.addRule(new FuzzyRule("temperature", "medium", "humidity", "medium", "medium"));  
  
 *// Правило 3: Если температура высокая и влажность низкая → максимальная скорость* fuzzySystem.addRule(new FuzzyRule("temperature", "high", "humidity", "low", "max"));  
  
 *// Правило 4: Если температура высокая и влажность средняя → быстрая скорость* fuzzySystem.addRule(new FuzzyRule("temperature", "high", "humidity", "medium", "fast"));  
  
 *// Правило 5: Если температура средняя и влажность низкая → средняя скорость* fuzzySystem.addRule(new FuzzyRule("temperature", "medium", "humidity", "low", "medium"));  
 }  
  
 */\*  
 Основной метод расчета скорости вентилятора  
 temperature - текущая температура  
 humidity - текущая влажность  
 return: рекомендуемая скорость вентилятора [0..100%]  
 \*/* public double calculateFanSpeed(double temperature, double humidity) {  
 Map<String, Double> inputs = new HashMap<>();  
 inputs.put("temperature", temperature);  
 inputs.put("humidity", humidity);  
  
 System.*out*.println("\n=== Начало процесса нечеткого вывода ===");  
  
 *// 1. Фаззификация входных данных* System.*out*.println("\n1. Фаззификация входных данных:");  
 printFuzzyStatus(temperature, humidity);  
  
 *// 2. Активация правил* Map<String, Map<String, Double>> fuzzifiedInputs = fuzzySystem.fuzzifyInputs(inputs);  
 Map<String, Double> outputActivations = fuzzySystem.activateRulesWithLogging(fuzzifiedInputs);  
  
 *// 3. Дефаззификация* System.*out*.println("\n3. Дефаззификация:");  
 double speed = CentroidDefuzzifier.*defuzzifyWithLogging*(fanSpeed, outputActivations, 0, 100);  
  
 System.*out*.println("\n=== Результат нечеткого вывода ===");  
 System.*out*.printf("Рекомендуемая скорость вентилятора: %.1f%%\n", speed);  
  
 return speed;  
 }  
  
 *//Печать текущих степеней принадлежности для входных значений* public void printFuzzyStatus(double temp, double hum) {  
 System.*out*.printf("Температура %.1f°C: low=%.2f, medium=%.2f, high=%.2f%n",  
 temp,  
 temperature.getMembership("low", temp),  
 temperature.getMembership("medium", temp),  
 temperature.getMembership("high", temp));  
  
 System.*out*.printf("Влажность %.1f%%: low=%.2f, medium=%.2f, high=%.2f%n",  
 hum,  
 humidity.getMembership("low", hum),  
 humidity.getMembership("medium", hum),  
 humidity.getMembership("high", hum));  
 }  
}

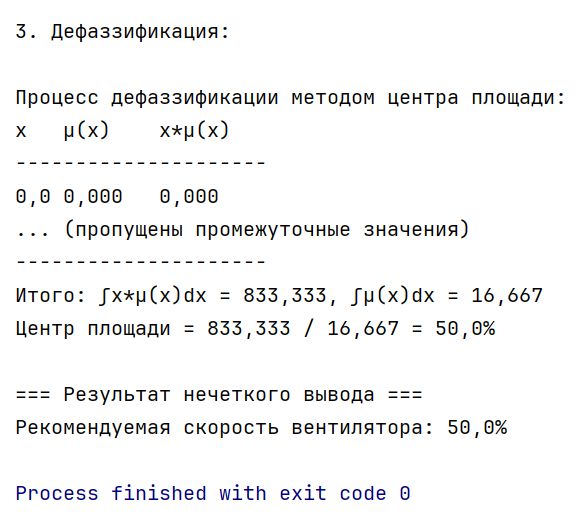
7. Main: Демонстрационный класс

import fuzzyLogicProgram.FanController;  
  
public class Main {  
 public static void main(String[] args) {  
 FanController controller = new FanController();  
  
 *// Пример 1: температура 27°C, влажность 45%* System.*out*.println("=== Пример 1: температура 27°C, влажность 45% ===");  
 double temp1 = 27.0;  
 double hum1 = 45.0;  
 double speed1 = controller.calculateFanSpeed(temp1, hum1);  
  
 */\*  
 // Пример 2: температура 32°C, влажность 60%  
 System.out.println("\n\n=== Пример 2: температура 32°C, влажность 60% ===");  
 double temp2 = 32.0;  
 double hum2 = 60.0;  
 double speed2 = controller.calculateFanSpeed(temp2, hum2);  
 \*/* }  
}

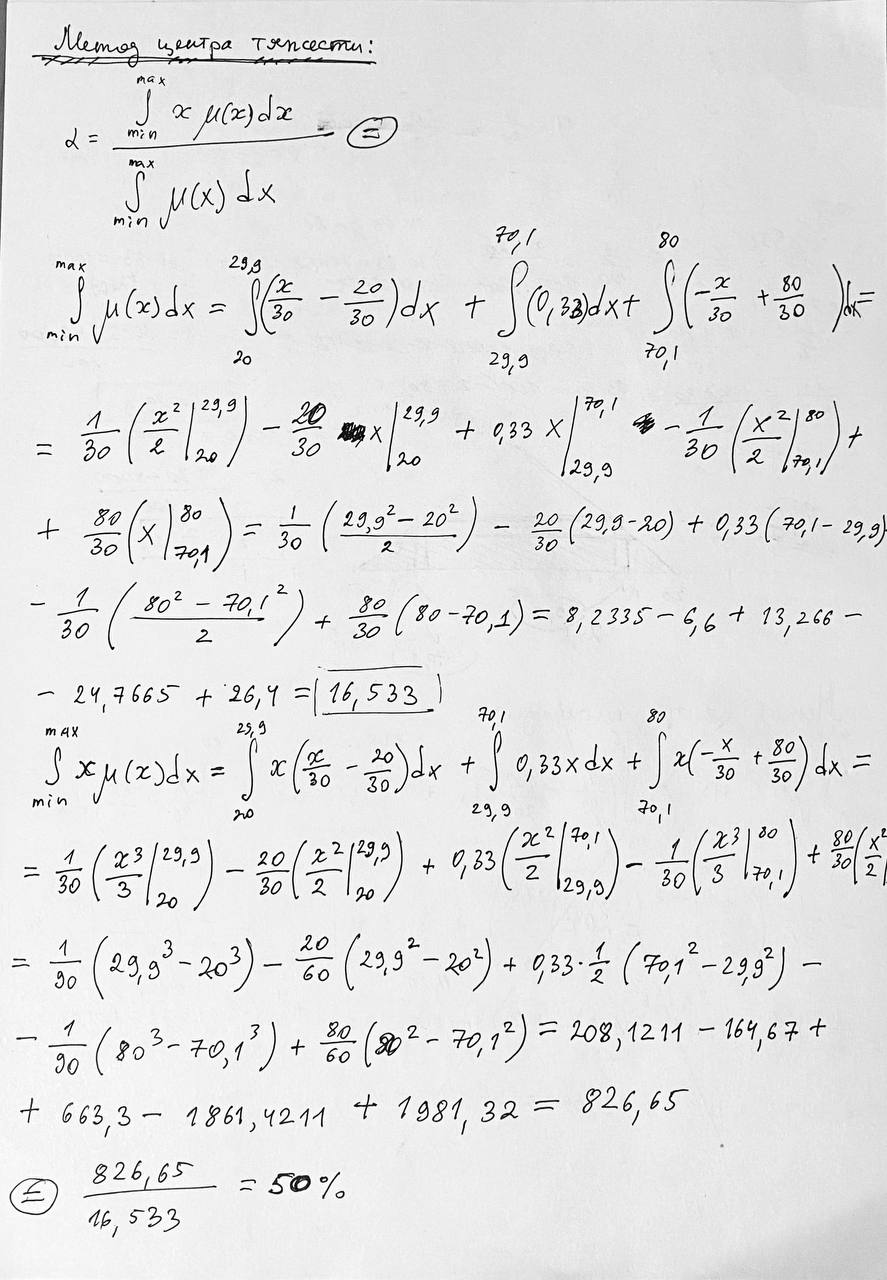
### Общая схема работы системы:

**Входные данные**: Четкие значения температуры и влажности  
**Фаззификация**: Преобразование в степени принадлежности к нечетким термам  
**Выполнение правил**: Активация соответствующих заключений  
**Агрегация**: Объединение результатов всех правил  
**Дефаззификация**: Преобразование нечеткого результата в четкое значение скорости  
**Вывод**: Рекомендуемая скорость вентилятора в процентах

Результат работы программы:  




Приложение:

****