Java Fuzzy Logic Engine
jFLe

von
Joe Trung

Overview

What is **Fuzzy Logic** or **FL** for short? What is fuzzy logic? Let's look to **Wikipedia** for that ominous fuzzy definition. Wikipedia says:

> Fuzzy logic is a form of many-valued logic in which the truth value of variables may be any real number between 0 and 1. It is employed to handle the concept of partial truth, where the truth value may range between completely true and completely false. By contrast, in Boolean logic, the truth values of variables may only be the integer values 0 or 1.
> The term fuzzy logic was introduced with the 1965 proposal of fuzzy set theory by mathematician **Lotfi Zadeh**. Fuzzy logic had, however, been studied since the 1920s, as infinite-valued logic -notably by **Łukasiewicz** and **Tarski**.
> Fuzzy logic is based on the observation that people make decisions based on imprecise and non-numerical information. Fuzzy models or fuzzy sets are mathematical means of representing vagueness and imprecise information (hence the term fuzzy). These models have the capability of recognising, representing, manipulating, interpreting, and using data and information that are vague and lack certainty.

So fuzzy logic is the concept of partially true or partially false. And this concept fits better with real life than discrete concepts where it is either true or false. Nothing in between. The computer is digital and has only two states that are either true (1) or false (0), while FL is a range of possible states that can serve as a result and that does not fit on the computer at all. In order to work with the computer, FL must be implemented in such a way that the partial states can be represented.

java Fuzzy Logic engine – jFLe

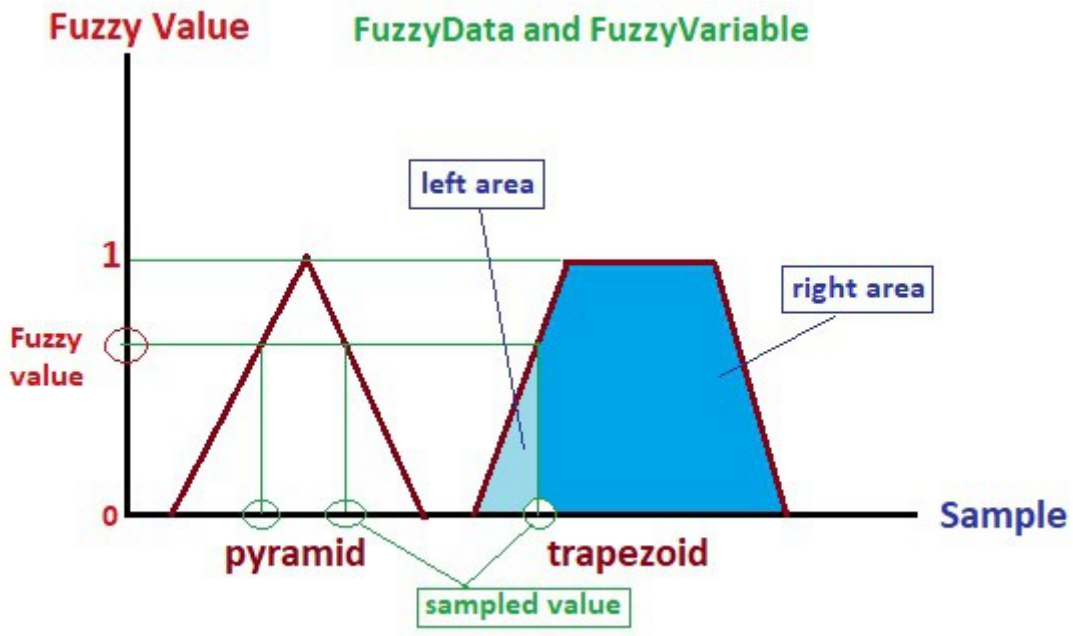
Java or any other object-oriented programming language is ideal for FL implementation. Due to the infinite possibility of states, the floating point value (float or double) can be used as a probability between 0...1 to represent the states. The FL state is implemented as a fuzzy variable and the probabilities are fuzzy data. Then we have:

- **FuzzyVariable** (**FV**) that holds various probability data (FD), FV is identified by an unique name FuzzyVariableName (**fvN**).
- **FuzzyData** (**FD**) contains a range of possible samples, FD is identified by an unique name FuzzyDataName (**fdN**).

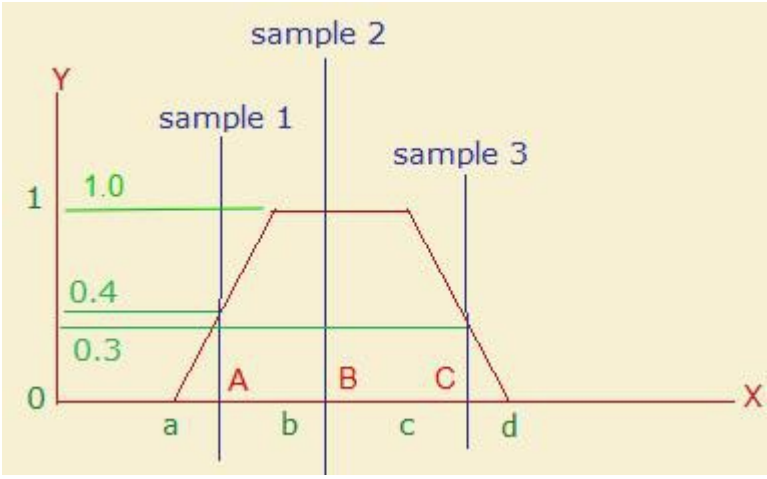
There are two different types of fuzzy data:

- The first has a range that increases from 0.0 to 1.0 and then decreases from 1.0 to 0.0. This shape is called a **pyramid** type.
- The second has a range that increases from 0.0 to 1.0, stays at 1.0 for a while and then decreases from 1.0 to 0.0. This shape is called a **trapezoid** type.

Each Pyramid (with 3 values) or Trapezoid (with 4 values) is a **Fuzzy Data Set** (range).



A pyramid is a special form of trapezoid where the left top value is also the right top value.Only the inner region of the pyramid or trapezoid represents the fuzzy states. All samples outside the inner region are undefined. A fuzzy value or **probability sample** always has two equal fuzzy states: to the left and to the right of this region. Let's take a closer look at a sample:



Suppose we have three samples, 1, 2 and 3. Sample 1 intersects the trapezoid on the left diagonal at 0.4 on the Y-axis, sample 2 intersects the horizontal vertex at 1.0 and sample 3 intersects the right diagonal at 0.3. The fuzzy variable containing these fuzzy data now has three samples: **A**, **B** and **C** on the **X-axis** with probability state **0.4**, **1.0** and **0.3** on the **Y-axis**. The trapezoid starts at **a** and ends at **d**. What lies beyond **a** or **d** is completely "false" and what lies between **a** and **d** is partially "true", which starts at **0.0**, goes to **1.0** and then decreases from **1.0** to **0.0**. This fuzzy dataset contains a left value **a**, a left-top value **b**, a right-top value **c**, and a right value **d**. The fuzzy value on the Y-axis is called the **fuzzified** sample, which is on the X-axis.

jFLe Implementation

A fuzzy variable can contain multiple FD sets. And each FD set can be either a pyramid or a trapezoid. The implementation of FD and FV in Java:

FuzzyData -FD

```
public class FuzzyData {
   /**
   Contructor, Pyramid FuzzyData
   @param name  String, FuzzyData name
   @param left  double, left value (must be less than top)
   @param top   double, Pyramid top
   @param right double, right value (must be greater than top)
   @exception Exception thrown by JAVA
   */
   public FuzzyData(String name, double left, double top, double right) throws Exception {
      init(name, left, top, top, right);
   }
   /**
   Contructor, trapezoid FuzzyData
   @param name  String, FuzzyData name
   @param left  double, left value (must be less than lTop)
   @param lTop  double, the Trapezoidal left top
   @param rTop  double, the Trapezoidal right top (must be greater than lTop)
   @param right double, right value (must be greater than rTop)
   */
   public FuzzyData(String name, double left, double lTop, double rTop, double right) throws Exception {
      init(name, left, lTop, rTop, right);
   }
   //
   private void init(String name, double left, double lTop, double rTop, double right) throws Exception {
      if (lTop < left || rTop < lTop || rTop > right)
          throw new Exception("Invalid Data: "+left+", "+lTop+", "+rTop+", "+right);
      ….
   }
   ...
```

FuzzyVariable -FV

```
public class FuzzyVariable {
   /**
   Constructor of FuzzyVariable (FV)
   @param name String, FV name
   */
   public FuzzyVariable(String name) {
      this.name = name;
      FD = new ArrayList<FuzzyData>(10);
      cache = new HashMap<String, FuzzyData>();
   }
   /**
   add a FuzzyData (FD) to this FV
   @param fd FuzzyData
   */
   public void add(FuzzyData fd) {
      FD.add(fd);
      cache.put(fd.getName(), fd);
      ….
   }
```

However, like Java, OOPL only provides queries with clear conditions and a distinct result, such as "**if... then... else...**". The result of such a query is that the then-block is executed if the conditions are met, otherwise the else-block is taken. And that is impossible for fuzzy logic. A fuzzy logic query can be partially “true” if something in the condition is met. And there is **no** alternative. Or in FL terms (A and C are FV, B and D are FD):

```
If A is B then C is D
```

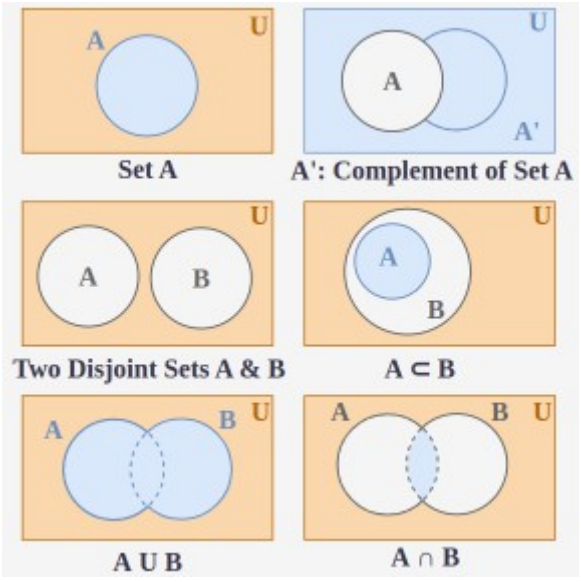
The **is** here signifies a range of possibilities where A partially agrees with B and so C gets this partial result (this possibility). To get the partial result, each sample must be "fuzzified". And this is done as follows:

```
/**
get fuzzified value of X
@param X double, sample Value
@return double, fuzzified value of X
*/
public double fuzzify(double X) {
    if (X <  left) return 0;
    if (X <  lTop) return (X-left)/(lTop-left);
    if (X <= rTop) return 1;
    if (X <  right) return (right-X)/(right-rTop);
    return 0;
}
```

And this fuzzified sample X, which now lies between **0.0** and **1.0**, is added as a point to the given FD **D**.

```
/**
add a Point to FuzzyData
@param d double, fuzzified sample
*/
public void addPoint(double d) {
    point += d;
    ++cnt;          // increment the number of added points
}
```

Furthermore, Fuzzy Logic is based on set theory. The implementations given represent only a subset of it (for an overview of set theory, click HERE).



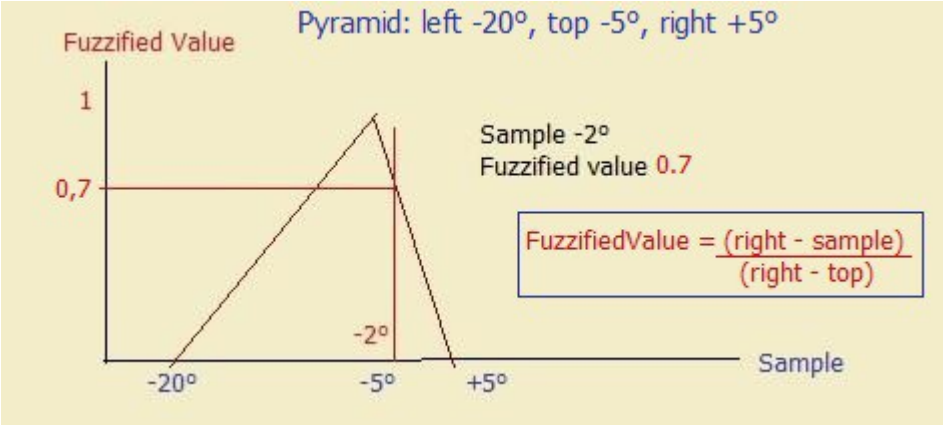
Let's take an example: A room has a temperature that can be hot, warm, fair, cool and cold. It depends on how people perceive the temperature. So a FV room could have 5 FDs, each of which has a range of values in Celcius (pyramid or trapezoid). Let's start defining FD cold, cool, fair, warm and hot for FV room:

```
declare room(cold(-20d, -5d, 5d), cool(5d, 10d, 19d), fair(19d, 20d, 21d), warm(21d, 23d, 25d), hot(24d, 30d, 40d))
```

We speak of cold when the temperature in the room is between -20° and +5° Celsius, cool when it is between +2° and +19°, etc. The air conditioning system functions as heating or cooling. So we have FV aircon and a scaling range between **-10** (cooling) and **+10** (heating). Let's define the FDs for FV AirCon: min, low, OFF, high, max.

```
declare AirCon(min(-10d, -6d, -4d), low(-4d, -2d, 0), OFF(-0.25d, 0, 0.25d), high(0d, 2d, 4d), max(4d, 6d, 10d))
```

Let's say we take a sample and it shows -2°



So we have a fuzzy query "if room is cold" and the result will be 0.8. Let us start a series of fuzzy queries for the room temperature, the results of which affect the FV AirCon.

```
if room is cold AirCon is max          // condition is met and 0.7 is added to point, counter = 1 of FV AirCon
if room is cool AirCon is high         // no sample for cool is found, 0 is added to point, counter = 2
if room is fair AirCon is OFF          // same … counter = 3
if room is warm AirCon is low          // same … counter = 4
if room is hot  AirCon is min          // same … counter = 5
```

FV aircon now has 0.7 points on 5 queries. How to get the correct value of accumulated points back from a FV? The reverse process of fuzzification is called "**defuzzification**" of the fuzzified points. It involves summing the accumulated points of all fuzzy data of the fuzzy variable to be calculated and calculating it as the fuzzified surface or centroid (center of gravity) of the pyramid or trapezoid. The defuzzified FV AirCon with 0.7 points and a counter of 5 is **6.53** and is realistic for an air conditioner with a temperature of **-2°** in the room (max heating of the pyramid range 4, 6, 10)

Usually it is sufficient to implement the fuzzy variable, the fuzzy data representation, the fuzzy query and the fuzzy script parsing. However, the context switching between the fuzzy implemented query and its underlying language can cause a large overhead that noticeably slows down the (computer) process. I have therefore built a simple interpreter that directly executes the basic statements like "do..while", "if..else..", etc., so that unnecessary context switching should be reduced as much as possible. And I call it the Java Fuzzy Logic Engine or **jFLe**.

The statements (case sensitive):

statement	Explanation and syntax (ref. To JAVA)
declare fvN(fdN1(..), .. fdNn())	declare a FuzzyVariable fvName consists of FuzzyData fdN1 with probability values in brackets. Example: FV with 3 pyramid FDs declare speed(slow(10, 30, 50), moderate(45, 80, 100), fast(95, 150, 200))
int **long** **double** **string**	like Java primitive int, long, double, String. Example: **int** a **string** x = "Hello" **double** d = 12.5
do .. **with** condition LABEL **do** .. **with** condition	do { … } while (condition). Example: **do** a = a+b **with** a < c or: **do** (a + b) **with** (a < c)
while condition..**endwhile** LABEL **while** condition **endwhile**	while (condition) { …. }. Example: **while** a < c a = a+b or **while** (a < c) ... **endwhile**
break **break** LABEL	similar to JAVA break
if fvName **is** fdName fuzzy	fuzzy query implementation and fuzzy, for example; **if** speed **is** fast speed **is** slowdown or **if** (speed **is** fast) speed **is** slowdown
if condition any instruction	e.g. **if** a **>** b a = b
if condition … **else** … **endif**	e.g. **if** a >b a = b **else** b = a **endif** (else and endif are optional)
set variable **=** ...	assign a value to variable where variable can be a primitive or FV
remove fvName(fdName)	remove a FD from FV
int a **=** 10 **print** "Hello "+a	assign 10 to a **print** "Hello "+a // on console: hello 10
clear fvName	reset all FuzzyData of FuzzyVariable (point = 0, cnt = 0)
reset fvName	reset sample to 0 of FuzzyVariable
// …..	comment from // to the end of the line
pause n	make a pause for n milliseconds
exit	terminate and exit the running FL script

In addition to the instructions, some reserved keywords

Keyword or symbole	examples
=	a = 10 fvName = 12 set fvName = 15
+ - * /	a = ((a+b)*c)/d
is, not, some, very	The fuzzy function of **not d** is the **1-d** (where **d** is a fuzzified value), **some** is a **d*d**, **very** is **Math.sqrt(d)** and **is** is an direct assignment to a FD of a FV.
this.java_method(..)	Invocation of a native Java method with/without parameters a = this.doSomething() this.save(a)
this:variable	Access to native Java variable. E.g. a = this:speed
&& || >= <= == > < ! & | not very some	Same meaning like java or set theory

jFLe is a package called "fuzzylogic" written in Java without any third party API. Pure Java standard APIs. And the package consists of 5 following APIs:

- **FuzzyFrame**: an abstract class
- **FuzzyEngine**: an extension of FuzzyFrame
- **FuzzyCluster**: a thread-safe class of FuzzyEngine group
- **FuzzyVariable**: the fuzzy specific variable
- **FuzzyData**: the fuzzy specific variable (pyramid or trapezoid)

To run jFLe, you need a plain text file called **fuzzy script** (extension **.txt**) with the syntax as in the above statements. Example:

The script **fuzzy.txt**

```
double y = 2
print "Y = "+y
declare fv1(fd1(10, 20, 30),fd2(10, 20, 30, 40), fd3(50, 60, 70))
declare fv2(fd1(10, 20, 30),fd2(10, 20, 30, 40), fd3(50, 60, 70))
set fv2 = 3
y = fv2
if (y == 3) y = 1+fv2
if (y == 4) print "1.true"
else print "1.false"
//
set fv2 = 20
if (fv2 is fd1) print "2.true"
if (fv2 is fd1)
  print "3.true"
  fv1 is fd1
  print "fv1 is set to "+fv1
else
  print "2.false"
  fv1 is fd1
  print "Again fv1 is reset to "+fv1
endif
//
print "Y = fv2 = "+y
fv2 = 13
print "new Y = fv2 = "+fv2
y = (y * 5 + 6)/7
print "final: ((y * 5) + 6)/7 = "+y+" Mission completed"
```

The Java app: **Fuzzy.java**

```
import java.util.*;
import fuzzylogic.FuzzyEngine;
public class Fuzzy {
  public static void main(String... args) throws Exception {
    new Fuzzy(args);
  }
  public Fuzzy(String... args) throws Exception {
    FuzzyEngine engine = new FuzzyEngine(this, "Fuzzy");
    if (args.length > 1) engine.setCentroid(true);
    engine.execute(args[0]);
  }
}
```

And the results on a console (here: Windows 10).

```
C:\JFX\FuzzyLogic\Test>java Fuzzy fuzzy.txt
Y = 2.00
1.true
2.true
3.true
fv1 is set to 19.99
Y = fv2 = 4.00
new Y = fv2 = 13.00
final: ((y * 5) + 6)/7 = 3.71 Mission completed

C:\JFX\FuzzyLogic\Test>
```

Explanation:

- The line **FuzzyEngine engine = new FuzzyEngine(this, "Test")** instantiates the JFL with 2 parameters: the bearer itself (this) and the app name (Test).
- The line **engine.execute(args[0])** starts interpreting the fuzzy script (args[0]) with syntax check and executes it line by line if no syntax error is found. Otherwise, an exception is thrown with the line where the syntax problem was found.

An exception example: let change the script line **y = fv2** to **a = fv2** (**a** is not declared). The exception **Exception in thread "main" java.lang.Exception: Unknown a operation at line:a = fv2 if (…** shows the line **a = fv2** and part of the next statement **if.** An unknown operation is assumed because **a** is neither declared as a primitive nor as FV/FD, so JFL assumes that **a** is an instruction it does not know.

```
C:\JFX\FuzzyLogic\Test>java Fuzzy fuzzy.txt
Y = 2.00
Exception in thread "main" java.lang.Exception: Unknown a operation at line:a =
fv2 if ( ...
        at fuzzylogic.FuzzyEngine.syntax(Unknown Source)
        at fuzzylogic.FuzzyEngine.execute(Unknown Source)
        at Fuzzy.<init>(Unknown Source)
        at Fuzzy.main(Unknown Source)
```

With the implemented **jFLe** instructions, it is not necessary to work with the provided API methods. The rule is simple:

- complex calculations, file IOs and GUI via direct method access of the Java app.
- simple calculations are performed locally in the script.

For example, the **setCentroid()** method overrides the default region surface calculation with centroid. As you can see from **fuzzy.java**, the **execute()** method is used here, which starts **jFLe**. Note: If **2** parameters are specified, **setCentroid()** is set to the centroid, but the result is not changed at all or only negligibly.

To get the full JAVA-style documentation, run **javadoc** on the sources. For example, the JAVA style documentation should be in the **doc** directory:

```
javadoc -d doc *.java
```

FuzzyFrame

- public void setPrinter(OutputStream printer) overrides default System.out with printer
- public abstract void execute(String script)

FuzzyEngine (a FuzzyFrame extension)

- public FuzzyEngine(Object obj, String objName), constructor with instantiated java obj as the main 'bearer' and its name objName
- public void resetEngine( ) implements the FuzzyFrame's resetEngine()
- public void execute(String script) implements the FuzzyFrame's execute()
- public void regClass(Object obj, String objName) registers instantiated java obj with objName
- public Object getClass(String objName) returns the instantiated obj of the given objName
- public Object removeClass(String objName) removes the registered obj of objName
- public void clearEngine() clears all FD points of all FVs
- public void setCentroid(boolean centroid) sets the defuzzification mode (default: surface) to centroid
- public void stop( ) stops engine where it stands
- public void resume( ) resumes the engine
- public Object execJava(String java) execute a Java method named java of a registered object (see regClass)

FuzzyCluster is intended for a multithreading environment with a cluster of FuzzyEngine

- public FuzzyCluster(String cName), constructor with Cluster name cName
- public void createFuzzyEngine(Object obj, String feID) creates a FE with the feID from instantiated obj
- public FuzzyEngine removeFuzzyEngine(String feID) removes FE with feID
- public FuzzyEngine getFuzzyEngine(String feID) gets FE with feID
- public void execute(String feID, String script) throws Exception runs FuzzyEngine with the feID and the script
- public double defuzzify(String feID, String fvName, boolean mode) defuzzifies FV of feID with the given mode
- public Object execJava(String feID, String java) runs a java method of feID
- public double fuzzyValue(String feID, String fvName) gets defuzzified value of FV of feID
- regObject(String feID, Object obj, String objName) registers feID an instantiated obj with objName
- public Object removeObject(String feID, String objName) throws Exception removes objName from feID
- public void reset( ) resets FuzzyCluster. All registered Fes are discarded
- public int size( ) returns the number of registered Fes

FuzzyData is a set of sample range in Pyramid or Trapezoid shape

- public FuzzyData(String name, double left, double top, double right) throws Exception, constructor for pyramid shape with an ID name
- public FuzzyData(String name, double left, double lTop, double rTop, double right) throws Exception, constructor for trapezoid shape with an ID name
- public boolean equals(FuzzyData FD) compares to another FD
- public double fuzzify(double X) fuzzifies sample X
- public void addPoint(double d) adds gained point from a Fuzzy Querry
- public boolean hasPoint() returns true if this FD has some points
- public double getPoint() returns the points
- public void reset() resets points and counter
- public double centroid() returns the centroid-calculated FD
- public String getName() returns the instantiated name of FD
- public double getMin() returns the min. range value of FD
- public double getMax() returns the max. range value of FD
- public String toString() returns FD short description

FuzzyVariable is the basis of fuzzy logic. FV can contain several FD

- public FuzzyVariable(String name), constructor FV with an ID 'name'
- public void add(FuzzyData fd) adds a FD
- public void set(FuzzyData fd) replaces a FD
- public FuzzyData remove(String fdName) removes a FD with its fdName
- public ArrayList<FuzzyData> getFDList() returns a FD list
- public boolean equals(FuzzyVariable FV) compares to another FV
- public double isFuzzy(String fdName) returns a fuzzified value of FD with fdName
- public void clear( ) clears all FDs
- public void reset( ) resets sample and clears all FDs
- public void addPoint(String fdName, double d) throws Exception adds points to FD with fdName
- public void setSample(double d) sets a sample
- public double getSample() returns a set sample
- public FuzzyData getFuzzyData(String fdName) returns FD with fdName
- public boolean contains(String fdName) true if FD with fdName exists
- public boolean contains(FuzzyData fd) true if FD exists
- public String getName() returns the ID of this FV
- public String toString() returns a brief FV description

Working with jFLe language Keywords

declare declares a FV with the given fvName that consists of a list of FD with fdName. Example: declare a FV with the name AirCon and 5 FDs with the names min, low, OFF, high and max.

```
declare AirCon(min(-10d, -6d, -4d), low(-4d, -2d, 0), OFF(-0.25d, 0, 0.25d), high(0d, 2d, 4d), max(4d, 6d, 10d))
```

Fuzzy Expression (FE) is a statement that contains FV, FD and fuzzy specific keywords such as **is** (similar to equal), **not** (complement), **some** (power of 2) and **very** (square of 2).

```
AirCon is max                    // this is a FuzzyExpression: AirCon is a  FV, max is a FD
```

if, if-then, if-else, if-else-endif is a questioning like if-else in another high-level language, with the only difference that the conditions are either **fuzzy** (FE) or **crisp** (a > b). Brackets are optional and can be used to improve readability. The keywords **then, else** and **endif** are optional and can be omitted. Note: With an **if** without else or endif, only ONE immediately following statement is executed (successful) or skipped (failed). A block of multiple statements is placed between if-else or if-endif or else-endif.

```
if room is cold AirCon is max              // fuzzy querry without 'then'
if room is cold then AirCon is max          // with 'then'
if (Obstacle is inFront)                   // fuzzy querry with block between if-else and else-endif
    Maneuver is right
    print "Maneuver:"+Maneuver
else
     print "else->Maneuver:"+Maneuver
endif
if (y == 4)                                // crisp query (conventional)
  print "1.true"
else
  print "1.false"                          // without endif
```

Primitive variables **int, long, double** and **String** are equals to Java primitives. Primitives like short, byte, float and char are not implemented.

```
double d = 13.6            // init to 13.6
string java = "java"       // init with "java"
```

Interoperations with java and script variables is in conjunction with **this:** (this with colon) for variables and **this.** (this with dot) for methods.

```
room = this:temp                    // variable
AirCon = this.adjust(temp)          // method with returned value as a sample for FV AirCon
```

Operations with **+, -, *, /, =, &&, ||, >, <, >=, <=, ==** and **!=** are similar to java operations.

```
if ((Distance is tooFarAhead || Distance is veryFarAhead) && Height is tooHigh) Speed is speedFast
Height = this:deltaY
y = (y * 5 + 6)/7
```

do...with, while...endwhile and **break** like java do..while, while {...} and break. A loop can have a label so that a break can refer to it. An **asterisk *** indicates an endless loop

```
While *                             // similar to java: while (true) { ...}
  …
  if a == b) break
  …
endwhile

LOOP while (x < 3 && x >= 0)        // while-Loop with label LOOP and crisp-conditions
  ...
  do
    if y == 1 break LOOP            // break the outer loop
    ...
  with obstacle is in front         // fuzzy condition
  …
endwhile
```

set is used to assign a FD to a FV, or to replace FD (if fdName is known) or to insert a new FD with fdName

```
set AirCon = 2              // assign sampe (value 2) to FV AirCon. Note: it can be used directly: Aircon = 2
set  AirCon(max(...))       // replace max of AirCon with a new max FD
```

remove removes FD with fdName from FV

```
remove AirCon(max)          // remove FD max from FV AirCon
```

print writes a string to console (default) or to a file (see method FuzzyFrame method **setPrinter**)

```
print "final: ((y * 5) + 6)/7 = "+y+" Mission completed"
```

pause, clear, reset and **exit** are control statements. The **pause** halts the engine for the specific time in millisecond, **clear** is **FV clear()** method, **clear *** (with asterisk) is similar to the method **clearEngine().** And **reset** is **FV reset(), exit** aborts the process and returns to the Java app.

JFL Syntax Summary

FuzzyLogic Keywords, Operatots, Syntax and Operations

All keywords (e.g. if, is, then, while, etc.) must be lowercase.

FuzzyVariables (FV) and FuzzyData (FD) MUST be registered or declared BEFORE Fuzzy Expressions (FE) or FuzzyScript (FS) can be executed. As in any programming language, variables names must be unique. The **else** and **endif** keywords are optional. If **else** or **endif** are omitted, **ONLY ONE** subsequent statement is skipped if the condition is met. In conditional expressions, parentheses **(** and **)** are optional, but must be used in pairs. The keyword **then** is optional: if FE **then** FE is the same with if FE FE.

The **this** keyword refers to the app that runs FuzzyEngine. Registered object classes can be referenced by FuzzyEngine.

Reference JavaVariables (or fields) with : (colon): **this:**JavaVariableName

Reference JavaMethods with . (dot): **this.**JavaMethodName(Parm1, …)

FuzzyVariable and FuzzyData and Primitives

```
declare FVname(FDName(a, b, c), FDName(a, b, c, d), ...)

int varName                    // default initialized to 0
long varName                   // 0
double varName                 // 0.0
string varName                 // null string
double varName = 13.6          // init to 13.6
string java = "java"           // init with "java"
```

Operation with JavaVariables

```
FV = this:temp                 // set FV sample with JavaVariable temp of JavaClass MyApp
this.temp = val                // set JavaVariable temp to val
this.temp + val                // add JavaVariable temp with val
this.temp - val                // sub JavaVariable temp from val
this.temp * val                // mul JavaVariable temp by val
this.temp / val                // div JavaVariable temp by val
```

Statements

```
if (FV is FD) FVr is Fdr       // brackets ( ) are optional

if (FV1 is FD1 or FV2 is FD2) FV3 is FD3
if (FV1 is FD1 || FV2 is FD2) FV3 is FD3

if (FV1 is FD1 and FV2 is FD2) FV3 is FD3
if (FV1 is FD1 && FV2 is FD2) FV3 is FD3

if (FV1 is FD1) or (FV2 is FD2) FV3 is FD3
if (FV1 is FD1 or FV2 is FD_2) and FV3 is FD3 FV4 is FD4
if ((FV1 is FD1 or FV2 is FD_2) and FV3 is FD3) FV4 is FD4

if (FV some FD) …              // fuzzySample**2 of FD
if (FV very FD) ...            // SQRT(fuzzySample) of FD
if (FV not  FD) ...            // (1-fuzzySample) of FD

if (...) FVr some FDr          // assign result**2 to FDr of FVr
if (...) FVr very FDr          // assign SQRT(result) to FDr of FVr
if (...) FVr not  FDr          // assign 1-result to FDr of Fvr
if (...)  statement            // (...) can be either Fuzzy or discrete. E.g. room is cold OR a >= b
                               // after 'then' only ONE assignment/statement is processed or ignored
                               // NOTE: after then NO else NOR endif is required.

If (…)                         // block
  ...                          // codes
else                           // 'else' can be omitted
  ...                          // codes
endif                          // endif is HERE optional

do                             // with or without label, e.g. LOOP do …. with (….)
  …
with (...)                     // (...) can be either Fuzzy or discrete (e.g. like a > b && c < d, etc.)
                               // for an ednless loop: do … with *
```

```
while (...)                          // with or without label, e.g. LOOP while (..)  …. endwhile
  ....                               // for an ednless loop: while *  … endwhile
endwhile

break                                // break do (inner) loop
break label                          // break do loop with label (e.g. break LOOP)

a = 0
a = b+c                             // simple math.operations with NON-FuzzyVariable
a = b-c                             // only with FV or JAVA Primitives: int, long, double.
a = b*c                             // NO byte, char, short, float, Array
a = b/c                             // embedded such as (a+b)*(c/d) is allowed
s = "Hi there"                      // assign a new content to s

a = ((a*b)+c)/d                     // embedded always with brackets

FV = val                            // FV sample = val
FV = FV + val                       // + FV sample with val
FV = FV - val                       // - FV sample from val
FV = FV * val                       // * FV sample by val
FV = FV / val                       // / FV sample by val

this:javaVar                        // access to java variable
this.javaMethod (..)                // invoke a Java method with/without parameters

set FV = value                      // set sample value to FV
set FV = FDName(a, b, c)            // add FD to FV. Overwite if existed
set FV = FDName(a, b, c, d)         // add FD to FV. Overwite if existed
                                    // create a FuzzyVariable with Fvnames
remove fvName(fdName)               // remove FuzzyData fdName from FuzzyVariable fvName
print "...."                        // print a string
print "..."+value                   // print String concatenated to value
print value                         // print a value (int, double, etc.)
pause 1000                          // take a pause of 1000 milliseconds
```

Operators and Keywords

```
+ - * / =                           // Math operation, Java:Field, Java.method
&& ||                               // and or
> == < >= <=                        // in conjuction with while and case only

do ...with                          // do loop
while...endwile                     // while loop
break                               // break while / do
exit                                // exit FuzzyScript back to JAVA
this                                // refer to the java bearer App (i.e. app that executes JFL)
reset                               // FV sample and all its FD points
clear                               // clear all FV-FuzzyData of FV
some                                // muliplication of 2 fuzzified values
very                                // SQRT of a fuzzified value
not                                 // complement of a fuzzified value (1-d)
&&
||
if
else                                // optional
endif                               // optional
is
set
declare
print
pause
```

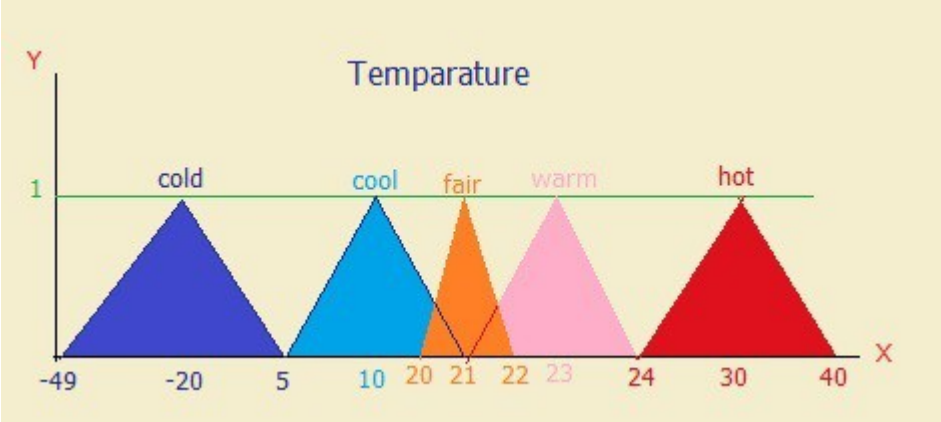
Other Java Fuzzy Logic Implementations

If you google Java Fuzzy Logic, you will find many JFL implementations that are totally different from mine. It is not my intention to comment or recommend some of them. For example, this **jFuzzyLogic** from **Github** or **SourceForge** can be freely download. A large and very complex JFL package. However, jFLe here is a FL engine that interpretes and runs independently along with the Java app, exchanging data and methods. And that is the difference between jFLe and jFuzzyLogic from Github and all other FL implementations.

An Implementation Example

Let's continue with the previous example: We have to regulate a house (or room) with an Air Conditioning sytem that has a scale between -10 (cooling) to +10 (heating). First, we have to determine the relationship and dependency of temparature (outside, inside) and the scale.

```
declare scale(min(-10d, -7d, -4d), low(-4d, -2d, 0), OFF(-0.25d, 0, 0.25d), high(0d, 2d, 4d), max(4d, 7d, 10d))
declare room(cold(-40d, -20d, 5d), cool(5d, 10d, 21d), fair(20d, 21d, 22d), warm(21d, 23d, 25d), hot(24d, 30d, 40d))
declare temp(cold(-40d, -20d, 5d), cool(5d, 10d, 21d), fair(20d, 21d, 22d), warm(21d, 23d, 25d), hot(24d, 30d, 40d))
```



We set the relationship and dependency between temparature and air conditioning system as following:

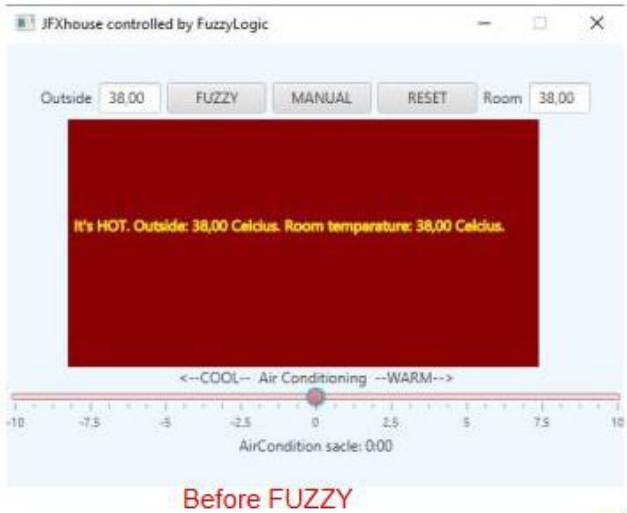
cold is between -40 to +5 degree	min is between -10 and -4
cool is between +5 and +21 degree	low is between -4 and 0
fair is between +20 and +22 degree	OFF is between -0.25 and +0.25
warm is between +21 and +24 degree	high is between +0 and +4
hot is between +24 and +40 degree	max is between +4 amd +10

With these specifications we can start designing a fuzzy script. The required samples for room and scale are taken from Java app. For simplicity, the scaling calculation is done by Java and the script simply calls it.

```
room = this:temp
scale = this:scale
while *
  if room is cold and scale is min temp is cold
  if room is cool and scale is min temp is cold
  if room is fair and scale is min temp is cool
  if room is warm and scale is min temp is cool
  if room is hot  and scale is min temp is fair

  if room is cold and scale is low temp is cold
  if room is cool and scale is low temp is cold
  if room is fair and scale is low temp is cool
  if room is warm and scale is low temp is fair
  if room is hot  and scale is low temp is warm

  if room is cold and scale is OFF temp is cold
  if room is cool and scale is OFF temp is cool
  if room is fair and scale is OFF temp is fair
  if room is warm and scale is OFF temp is warm
  if room is hot  and scale is OFF temp is hot

  if room is cold and scale is high temp is cool
  if room is cool and scale is high temp is fair
  if room is fair and scale is high temp is hot
  if room is warm and scale is high temp is hot
  if room is hot  and scale is high temp is hot

  if room is cold and scale is max temp is warm
  if room is cool and scale is max temp is hot
  if room is fair and scale is max temp is hot
  if room is warm and scale is max temp is hot
  if room is hot  and scale is max temp is hot

  // save temp and invoke Java adjust()
  scale = this.adjust(temp)
  room = this:temp
  if room is fair break
endwhile
```

```
public class JFXhouse extends Application {

  public void start(Stage stage) {
    stage.setTitle("JFXhouse controlled by FuzzyLogic");
    ran = new java.util.Random();
    // FuzzyLogic.....................
    eng = new FuzzyEngine(this, "JFXhouse");
    //
    Canvas canvas = new Canvas(400, 200);
    ...
    FUZZY.setOnAction(e -> {
      try {
        FUZZY.setDisable(true);
        MAN.setDisable(true);
        eng.execute("script/thishouse.txt");
        ...
      } catch (Exception ex) {
        ex.printStackTrace();
        stop();
      }
    });
    ...
  }
  // adjust the AirConditioning scale
  private double adjust(double d) {
    // Cooling: -40 .. +20: 10/60 = 0.1667
    // Heating: +20 .. +40: 10/20 = 0.5
    temp = d; // set temparature
    if (temp < 20) scale += 0.16667;
    else if (temp > 22) scale -= 0.5;
    return scale;
  }
  …
  private double temp, scale, outside;
}
```

And here is the JAVA -FX GUI app for our example: JFXhouse



Before FUZZY

After FUZZY

on Console

```
C:\JFX\FuzzyLogic\examples>java JFXhouse
Before of FUZZY control.
Outside 38.00 C. Room Temparature 38.00C. Aircon scale:0.00
End of FUZZY control:
Outside 38.00 C. Room Temparature 21.88 C. Aircon scale:-2.50
```

Joe T. Schwarz