If you thought that things like Artificial Intelligence or logical programming are all dead and buried in the 80s and have no relevance to our enterprise projects, think again. Drools is a Java framework that implements a form of AI called rule-based Expert System that, it might not win you Jeopardy ([http://en.wikipedia.org/wiki/Watson\_%28computer%29http://en.wikipedia.org/wiki/Watson\_%28computer%29](http://en.wikipedia.org/wiki/Watson_(computer)http://en.wikipedia.org/wiki/Watson_(computer))) but it is an open-source project that can help you quickly process data according to large sets of business rules and it will allow you to define those rules in a readable, user-friendly way. When looking at a rule, it is pretty clear of what it is about to both a developer with little business domain knowledge and to a business analyst:

<funky example of dslr>

**1. Some theoretical background**

Expert Systems are computer systems that make decisions like a human expert would, based on a method of knowledge representing (which forms their so called knowledge base) they infer conclusions. This is different from conventional programming because it doesn't work by following a procedure but instead it tries to mimic human reasoning about knowledge. Drools is a Rule Engine that uses the rule-based approach to implement an Expert System. A Rule engine is any system that uses rules, in any form, that can be applied to data to produce outcomes.

To bring back some of the traumas of the August exams in college, the official documentation adds that Drools is more precisely classified as a Production Rule System, a concept in Formal Grammars (<http://en.wikipedia.org/wiki/Formal_grammar>).

In rulebased systems, knowledge is represented in the form of if-then rules. For example, the following rule could be part of such a system:

IF Person wants to buy a house

Person does not have enough money for a house

THEN Person goes to bank for a loan

To actually trigger this rule, we will need a Person object/fact matching the conditions of the rule. We need to provide our rules with a number of facts where they can work upon.

The process of that decides weather each fact satisfies the Rules is called Pattern Matching, and is performed by the so called Inference Engine. There are several algorithms that can be used by an Inference Engine for pattern matching, Drools implements the Rete algorithm (some details later on). If a fact satisfies more than one rule, the matched rules are said to be in conflict and it becoms the job of a component called Agenda to decide the order in which those rules will be executed.

The Rules are stored in the Production Memory and the facts that the Inference Engine matches against are kept in the Working Memory.

**2. The Drools language**

Say we want to include in a Java project a simple expert system that would decide weather a person is eligible for a loan product or not. First, we'll need two POJOs:

package droolstest.loans;

public class Person {

private String name;

private Long salary;

private boolean loyalCustomer;

…

}

package droolstest.loans;

public class LoanFormula {

private Long minSalary;

private String name;

…

}

Next comes the fun part, writing the rules. Drools has a "native" rule language. This syntax is very light in terms of punctuation, and supports natural and domain specific languages using "expanders" that allow the language to resemble natural language for your problem domain.

*2.1 Drools native rule format - DRL*

Here's how some basic rules would look like using first just the native format:

1. package droolstest.loans.rules
2. import droolstest.loans.Person
3. import droolstest.loans.LoanFormula
4. rule "loyal\_customer\_rule"
5. when
6. Person(loyalCustomer == true, $personName: name)
7. $lf: LoanFormula()
8. then
9. System.out.println("Loan " + $lf.getName()
10. + " possible because person " + $personName + " is a loyal customer.");
11. end
12. rule "salary\_rule"
13. when
14. Person($salary: salary, $personName: name)
15. $lf: LoanFormula(minSalary <= $salary)
16. then
17. System.out.println("Loan "" + $lf.getName()
18. + " possible because person " + $personName + " has a satisfactory salary.");
19. end

This would be in a text file called, say LoanExpert.drl. Drl, short for Drools Rule Language, is the standard extension for Drools native language files.

First is the package definition. A package represents a namespace, which ideally is kept unique for a given grouping of rules. The package name itself is the namespace, and is not related to files or folders in any way. We'll place it under {project\_home}/src/droolstest/loans/rules/. Next come the imports, the Java types we reference.

The two most interesting parts of a rule are obviously the when and the then part. The when part, which is called the left hand side(LHS) of the rule, contains the conditions that need to be fulfilled in order for the then part, called the right hand side(RHS) of the rule, to execute. Writing a rule, the mindset is a little similar to writing SQL statements.

The first rule, called “loyal\_customer\_rule”, is supposed to make sure that if a person is a loyal customer, all loan formulas will be available to him. The “loyal\_customer\_rule” will match for every possible combination of a person with his property loyalCustomer set to true, and any loanformula. Line 8 does three things: it matches any instance of Person from the Working Memory, that has the loyalCustomer flag true and then it binds the variable personName to the value of that person's name. The prefixed dollar symbol ($) is just a convention but it can be useful in complex rules where it helps to easily differentiate between variables and fields.