**1.)Why We need Drools rule based engine?**

ans-We use Drools rule where any language is not enough to develop complex business rule.In this case we use Drools business rule engine to develop complex business rule.It is also use for seprate the Business logic to application layer.

In short,

### Drools is a library and rules engine in Java that lets you add business rules (logic) separate from other code in the system.

**2.What is Rule Engine?**

ans-It is place where we are evaluating our business rule.

This engine is already having some predefine rule or knowledge and based on that it draws some conclusion.

Rule Engine is pluggable piece of software component that seprate the business rule from application code.

Rule:--

Rule

when

<Condition>

then

<take action>

end

This rule is written in .drl file(Domain rule language).

for real time example of Miscrosoft outlook where we are imposing the Rule.

like

Rule 1(Vicky)

when

is mail coming from vicky

then

move it to junk folder

end

**3.)what is Drools?**

ans-Drools is Business Logic integration integration platform(BLIP) written in java by extending and implementing Rete pattern-matching algorithm.

It is open source project but there is community and enterprised version of this project is available.

Drools helps to segregating the Business rule from application layer.This rule is written in .drl file(Domain rule langauge).

Rule or Logic ,in our case is piece of knowledge an expressed as given below.

Rule

when

<Condition>

then

<take action>

end

Drools is divided into two parts.

Authoring:-Authoring alsways involve to creation of Rule file (.drl) which contents the syntax for the rule .This syntax can be parse by the parser and will generate appropriate structure for the Rule.

Runtime:-Creation of working memory and Handling activation which means that loading the rule in engine and how to execute it will be taken care by the Runtime.

**4.)Problem with the Traditional Appraoch.**

ans-Drools is extensively used in BFSI(Business Financial services and insurance).The primary reason is keep changing the requirement and adding new rule make the BFSI field volatile in nature.

**5.)Advantages of Drools.**

ans-

Easier to understand

Rules are easier to understand for a business analyst or a new developer than a program written in Java or other imperative-style language.

Flexibility:

It deals better with changes to the requirements or changes to the data model. Changing or rewriting an application is never an easy task. However, thanks to the formalism that rules bring, it is much easier to change rules than to change a Java program.

Reasonable performance

Thanks to the Rete algorithm that is behind Drools; in theory, the performance of the system doesn't depend on the number of rules. With every release of Drools, the performance of the engine is getting better by adding various optimizations such as Rete node sharing, node indexing, parallel execution, and so on. All this benefits new as well as old rules.

Reusability:

The rules are kept in one place (separation of business logic from the rest of the system), which means easier reusability. For example, imagine you've written some validation rules for your application and later on there is a need to do some batch imports of data, so you could simply reuse the validation rules in your batch import application.

Redeploying

It is possible to change/redeploy rules and processes without even stopping the whole application.

**6.)When not to use a Rule Engine?**

Ans-here are a lot of advantages of Rule Engine but that doesn't mean it can be a solution to all the application. It's not a Panacea for business layer so should be used wisely.

Below are the points to ponder:

If your project is one short effort and never be use again or maintained over time.

if rule contain one rule only but we should think again because every system grows in complexity overtime.

Whereas if you are about to use a Trading Application where you have new regulators on boarded every 3 months and new laws/rules regulating your application and daily new rules being told to implement and where changes are very frequent, its advisable to use Drools.

**7.)Different Terms used in Drools**

a.)Rules

b.)Facts

c.)Session

d.)KnowledgeBase

e.)Agenda

f.)Activations

Rules :

Rules are the heart of the Rule Engine. In rules you specify the condition and then the execution part of the condition. Below is the syntax of the rule in drools:

rule:"My First Rule"

when

#condition

then

#action

end

Facts :

The objects on which the rules are fired are known as Facts. It can be a simple Java POJO or a complex object. Java object is synonyms to facts in Drools

Session :

KnowledgeSession is core compenent in drools where rule will be fired.KnowledgeSession hold the rule and other resources.

KnowledgeSession will be created through KnowledgeBase.

To Trigger the rule ,fact will be inserted into session and if condition will be matched then particular rule will be fired.

#Session is two type

1.Stateless KnowledgeSession:-

A Stateless knowledge session won't track the changes in the facts while the rules are being fired. In simple words once we have inserted all the facts in the session and triggered all the rules then the rules cannot be fired on the updated facts. To draw a parallel, it would be similar like a web application where we are not tracking the session.

2.Statefull KnowledgeSession:-

In a Stateful knowledge session, once all the facts are inserted and rules are triggered and one rule is updating the fact and post the updation of fact another rule can also trigger as the fact is still in the session. Its similar where in a web application until the session is active all the items you add to the cart are visible.

KnowledgeBase :

This is an interface that manages a collection of rules, processes, and internal types. In Drools these are commonly referred to as knowledge definitions or knowledge.

Agenda :

It's a logical concept. The agenda is the logical place where rules are waiting to be fired

Activations :

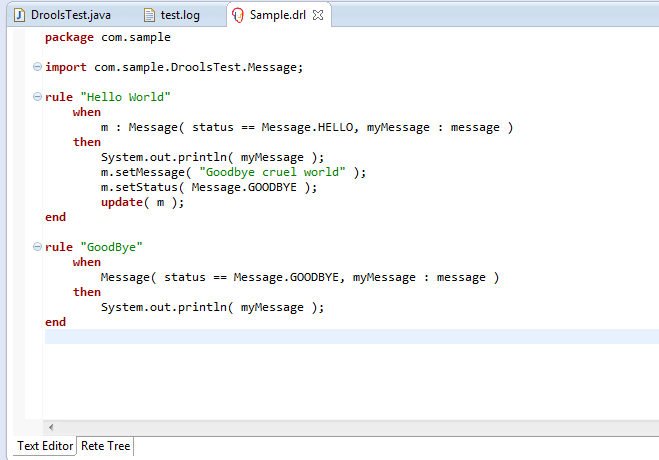
The then part of the rule. Activations are placed in the agenda where the appropriate rule is fired.

**8.)Rule writing or Syntax basics**

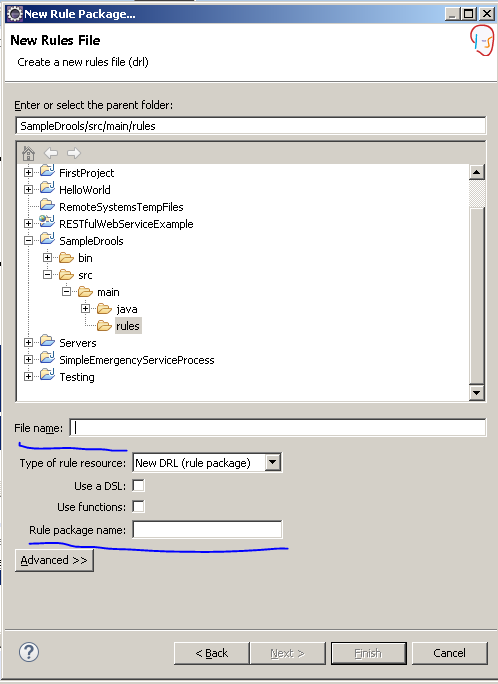
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Rule writing or Syntax basics**

If you see the default rule which is written in the **FirstDroolsProgram** project(Sample.drl), there are a lot of keywords used. Let us explain them one by one now :



1. **Package :** Every Rule starts with a package name. The package acts as a namespace for Rules. Package name has to be defined while creating the Rule resource. Rule names within a package must be unique. Packages in Rules are similar to packages in Java. When you create a new Rule by **New -> Rule Resource ->** then in the below window you need to mention the Rule name as well as package name.



1. **Import statement :** Import statements are also similar to Java. All the FACTS classes and any helper classes needs to be imported for the rules to work on. For ex/- **com.sample.DroolsTest.Message;** in the above example.
2. **Rule Definition :** The Rule definition consists of some drools specific keywords. Post you have added the plugins you can see the drools specific keywords in red color. In the above example, **rule-when-then-end** are the different keywords.

**rule** keyword must be followed by a logical rule name.

**when** keyword must be followed by some condition.

**then** keyword must have the consequence part if the when condition is met.

There are lot of other keywords which will be covered in further sections.

1. **Global variables :** With global you define **global variables**. They are used to make application objects available to the rules. Typically, they are used to provide data or services that the rules use, especially application services used in rule consequences, and to return data from the rules, like logs or values added in rule consequences, or for the rules to interact with the application.

Now let us walk you through the terms used in the Java file used to load the drools and execute the rules.

**9.)Rule Syntax**

The Drools Rule Resource(DRL) has a different syntax and covering the syntax in the below section :

Conditions in Rules :

A rule can contain many conditions/patterns:

\* Student(id == 001)

\* Employee(name == "Vivek")

The above conditions checks if Student id is 1 and the Employee name is Vivek.

Variable in Rules :

A variable name in Drools starts with a $ dollar symbol.

$student : Student()

$student is the variable for Student() class.

This is similar to Student student = new Student() in java. Drools can work with all the native Java types and even Enum.

Comments in Rules:

In Drools 5.x, # or // can be used as a single line comments

For Multi Line comments:

/\*

Another line

\*/

Functions in Rules:

Functions are a convenience feature. They can be used in conditions and consequences. If there is any modification to be done in the then part of the rule, a normal static java helper function can be called.

function double calculateAreaofSquare(double value)

{

return value \* value;

}

Dialect:

Dialect specifies the syntax used in any code expression that is in a condition or a consequence. The default value is Java. Drools currently supports one more dialect called mvel.

The default dialect can be specified at the package level as follows:

package org.mycompany.somePackage

dialect "mvel"

MVEL Dialect:

mvel is an expression language for Java-based applications. mvel supports field and method/getter access. It is based on Java syntax.

The no-loop attribute

This attribute informs the rule engine that a rule should be activated only once per matched facts. If there is a generic condition in the rule, then this attribute should be used to avoid the infinite loop scenario.

Salience:

Salience is a very important feature of Rule Syntax.

It is used by the conflict resolution strategy to decide which rule to fire first.If there are two rules and both have the conditions are met, then salience is used to determine the order of rule firing. It has one attribute, which takes any expression that returns a number of type int (positive as well as negative numbers are valid). The higher the value, the more likely a rule will be picked up by the conflict resolution strategy to fire.

salience ($account.balance \* 5)

salience 100

The default salience value is 0. We should keep this in mind when assigning salience values to some rules only. There are a lot of other features/parameters in the Rule Syntax but we have covered the important ones.

**10.) Rule Consequence Keywords**

Rule Consequence Keywords means the keywords which are used in the "then" part of the rule.

1. Modify: The attributes of the fact can be modified in the then part of the Rule.

2. Insert: Based on some condition if true, one can insert a new fact into the current session of the Rule Engine.

3. Retract: If a particular condition is true in a Rule and you don't want to act anything else on that fact, you can retract the particular fact from the Rule Engine.

Note : It is considered very bad practise to have conditional logic(if statements) within rule consequence. Most of time, a new rule should be created.

## 11.) How to call a external function from a .drl file.

Let us demonstrate how to call a static function from a java file within your DRL file. Creating a class Utility.java in the same package com.sample

public class Utility {

public static void **writeHello**(String name) {

System.out.println(**"HELLO"** + *name* + **"!!!!!!"**);

}

}

We then add the import statement in our drl file to call the writeHello method from our drl file. Again we have added the syntax in the same drl file used

#### Use case for this scenario :

* There are lot of times where you need to do calculation with respect to the parameters of your **Object** which you are using in your **DRL's** or for that matter you need to check whether the particular variable is a part of some **static data** or a **static configuration** and based on that decision you want to execute your rule, so at such places it's advisable to call a **external function**either in your DRL file or in your Utility class so that processing can be done in those functions and the rules can act on the output of your function.
* Primarily Drools is meant for the **business layer** i.e to evaluate what to do next if a particular condition is met and hence the processing part in case if needed for specific rules can be done in function and hence the provision given.
* Another example is that Java is a **Object oriented language** but it still lets you write **static/helper**functions to do some calculations and use static method without initiating any object. It is against the object oriented programming concept but still we can directly call a method without creating a object, so similarly Drools also provides that feature.

## 12.) How to Debug a Drools Project

There are different ways to debug a drools project.

#### Writing a Utility class to let you know which rules are being triggered or fired.

With this approach we can check what all rules are getting triggered in our drools project.

Here is our **Utility Class : Helper.java**

package com.sample;

import *org.drools.spi.KnowledgeHelper*;

public class **Helper** {

public static void **help**(final KnowledgeHelper drools, final String message){

System.out.println(message);

System.out.println(**"\nrule triggered: "** + drools.getRule().getName());

}

public static void **helper**(final KnowledgeHelper drools){

System.out.println(**"\nrule triggered: "** + drools.getRule().getName());

}

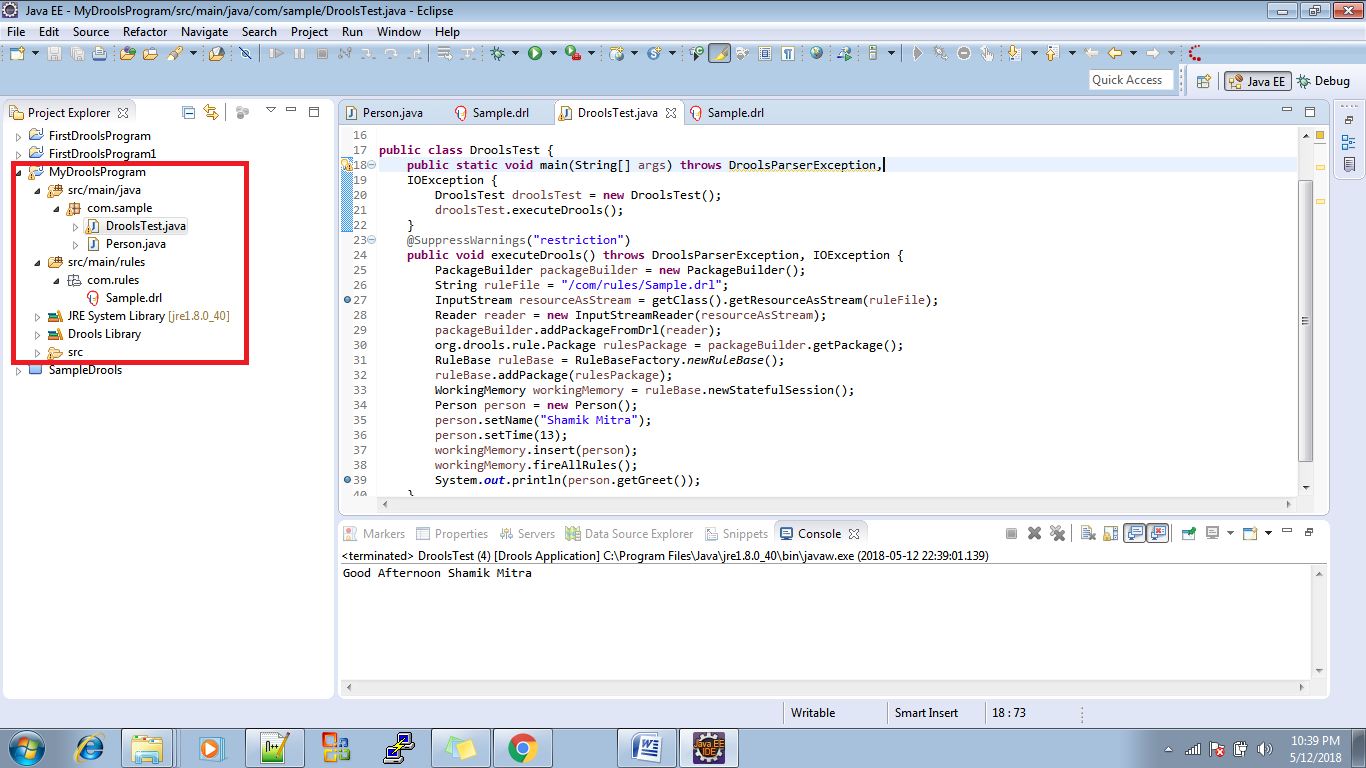
}

The first method help prints the rule triggered along with some extra info which you can pass as **String** via the drl file.

The second rule helper prints whether the particular rule was triggered or not.

**One Complete example to understand the Drools using Eclipse**

In this example, we will greet a person based on current time. We will define the rules in Drools files. Drool will load these rules and fire on the incoming data.



we create three rules: “Good Morning”, “Good Afternoon “ and “Good Night.” In the **When** section, we check the current time based on the Person POJO’s time property. In the **Then** section, we set the greeting messages accordingly

Sample.drl

**package** com.sample

**rule** "Good Morning"

**when**

person: Person(time >= 0, time < 12)

**then**

person.setGreet("Good Morning " + person.getName());

**end**

**rule** "Good Afternoon"

**when**

person: Person(time >= 12, time < 16)

**then**

person.setGreet("Good Afternoon " + person.getName());

**end**

**rule** "Good Night"

**when**

person: Person(time >= 16, time <= 24)

**then**

person.setGreet("Good Night " + person.getName());

**end**

**Step 2:**Create Person POJO class.

**package** com.sample;

/\*\*

\* This is a sample class to launch a rule.

\*/

**public** **class** Person {

**private** String name;

**private** **int** time;

**private** String greet;

**public** String getGreet() {

**return** greet;

}

**public** **void** setGreet(String greet) {

**this**.greet = greet;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getTime() {

**return** time;

}

**public** **void** setTime(**int** time) {

**this**.time = time;

}

}

**Step 3:** We create a class named DroolsTest.java.

3a. Load the rule file (i.e., droolsTest.drl) by using InputStream.

3b. Create a package using the above rule and add them into drools PackageBuilder.

3c. Create a RuleBase by using the above Package. Rulebase is the same as Sessionfactory in Hibernate; it is costly.

3d. Create a working memory from this RuleBase. It is same as Session class in Hibernate. This working memory manages the rules and incoming data. Apply the rules on the data.

3e. Add incoming data into working memory. Here, we create a Person Object and add it into Working Memory

3f. Fire all rules.

DroolsTest.java should look like the following:

package com.sample;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.Reader;

import org.drools.RuleBase;

import org.drools.RuleBaseFactory;

import org.drools.WorkingMemory;

import org.drools.compiler.DroolsParserException;

import org.drools.compiler.PackageBuilder;

public class DroolsTest {

public static void main(String[] args) throws DroolsParserException,

IOException {

DroolsTest droolsTest = new DroolsTest();

droolsTest.executeDrools();

}

@SuppressWarnings("restriction")

public void executeDrools() throws DroolsParserException, IOException {

PackageBuilder packageBuilder = new PackageBuilder();

String ruleFile = "/com/rules/Sample.drl";

InputStream resourceAsStream = getClass().getResourceAsStream(ruleFile);

Reader reader = new InputStreamReader(resourceAsStream);

packageBuilder.addPackageFromDrl(reader);

org.drools.rule.Package rulesPackage = packageBuilder.getPackage();

RuleBase ruleBase = RuleBaseFactory.newRuleBase();

ruleBase.addPackage(rulesPackage);

WorkingMemory workingMemory = ruleBase.newStatefulSession();

Person person = new Person();

person.setName("Shamik Mitra");

person.setTime(13);

workingMemory.insert(person);

workingMemory.fireAllRules();

System.out.println(person.getGreet());

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2.)Example simple Hello world

**package** com.sample

**import** com.sample.DroolsTest.Message;

**rule** "Hello World"

**when**

m : Message( status == Message.HELLO, myMessage : message )

**then**

System.out.println( myMessage );

m.setMessage( "Goodbye cruel world" );

m.setStatus( Message.GOODBYE );

**update**( m );

**end**

**rule** "GoodBye"

**when**

Message( status == Message.GOODBYE, myMessage : message )

**then**

System.out.println( myMessage );

**end**

---\*

package com.sample;

import org.drools.KnowledgeBase;

import org.drools.KnowledgeBaseFactory;

import org.drools.builder.KnowledgeBuilder;

import org.drools.builder.KnowledgeBuilderFactory;

import org.drools.builder.KnowledgeBuilderError;

import org.drools.builder.KnowledgeBuilderErrors;

import org.drools.builder.ResourceType;

import org.drools.io.ResourceFactory;

import org.drools.logger.KnowledgeRuntimeLogger;

import org.drools.logger.KnowledgeRuntimeLoggerFactory;

import org.drools.runtime.StatefulKnowledgeSession;

/\*\*

\* This is a sample class to launch a rule.

\*/

public class DroolsTest {

public static final void main(String[] args) {

try {

// load up the knowledge base

KnowledgeBase kbase = readKnowledgeBase();

StatefulKnowledgeSession ksession = kbase.newStatefulKnowledgeSession();

KnowledgeRuntimeLogger logger = KnowledgeRuntimeLoggerFactory.newFileLogger(ksession, "test");

// go !

Message message = new Message();

message.setMessage("Hello World");

message.setStatus(Message.HELLO);

ksession.insert(message);

ksession.fireAllRules();

logger.close();

} catch (Throwable t) {

t.printStackTrace();

}

}

private static KnowledgeBase readKnowledgeBase() throws Exception {

KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();

kbuilder.add(ResourceFactory.newClassPathResource("com/rules/Sample.drl"), ResourceType.DRL);

KnowledgeBuilderErrors errors = kbuilder.getErrors();

if (errors.size() > 0) {

for (KnowledgeBuilderError error: errors) {

System.err.println(error);

}

throw new IllegalArgumentException("Could not parse knowledge.");

}

KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();

kbase.addKnowledgePackages(kbuilder.getKnowledgePackages());

return kbase;

}

public static class Message {

public static final int HELLO = 0;

public static final int GOODBYE = 1;

private String message;

private int status;

public String getMessage() {

return this.message;

}

public void setMessage(String message) {

this.message = message;

}

public int getStatus() {

return this.status;

}

public void setStatus(int status) {

this.status = status;

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

References:-

https://www.studytonight.com/drools/add-drools-plugin.php

http://www.drools.org/

<https://www.slf4j.org/download.html>

<https://dzone.com/articles/learn-drools-part-1-1>

<http://www.javainuse.com/>