**Drools Concepts:**

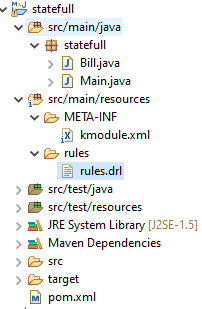
**Activation Group** a part of a rule, placed on the Agenda where the appropriate rule is fired. ( if else block )

**Salience** is a keyword in the \*.drl file that is assigned a positive or negative number. The more positive the number the higher the rule priority.

**Decision Table** a tabular representation of information.

**Cross Product** rules are called by the cross product of rules. One rule should not call another rule.

A **stateful** Session example:



Bill.java model code



rules.drl



kmodule.xml



Main.java driver code

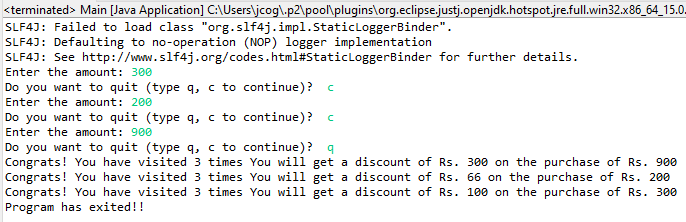


POM file

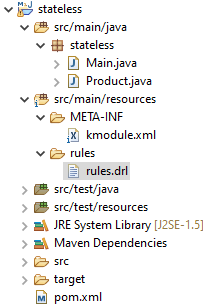


Drools remembers the count of inputs and divides by that count.

Therefore, this is a **stateful application**!!



A **stateless** Session example:



A stateless driver **does not** use fireAllRules()

You just use an execute()

StatelessKieSession kSession = kContainer.newStatelessKieSession("ksession-rule");

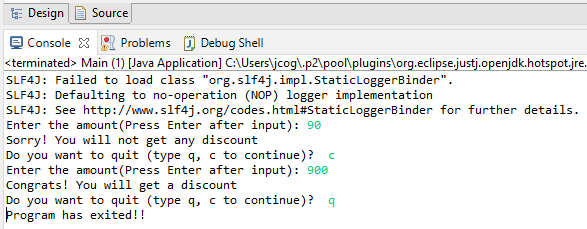
<ksession name=*"ksession-rule"*

type=*"stateless"* default=*"true"* clockType=*"realtime"*/>

No, dispose() commands are necessary, cause it is a stateless session

Drools does not remember the inputs.

Therefore, this is a **stateless application**!!



Product.java



kmodule.xml



Main.java



rules.drl

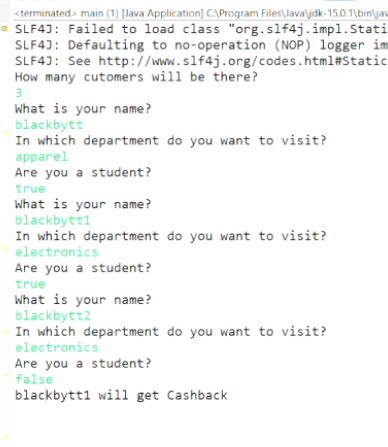


pom.xml



**Salience:** Controlling Priority

Salience Example:



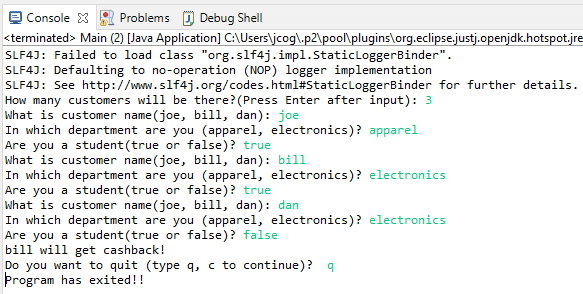
Joe, apparel, true

Bill, electronics, true will get Cashback

Dan, electronics, false

So, electronic products and bought by students will get Cashback

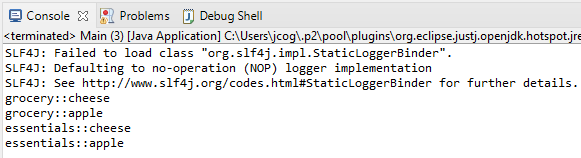
Salient Example:



2 X 2 = 4

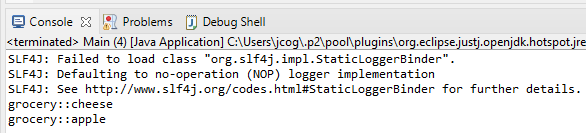
**Cross Product** without Filtering Example:



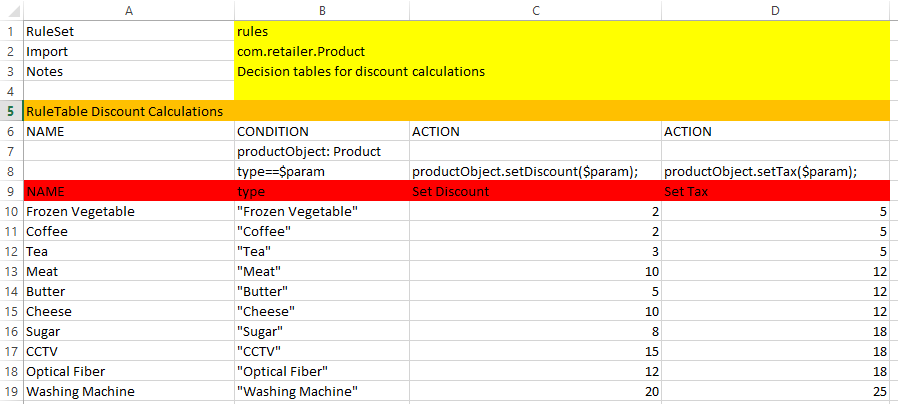


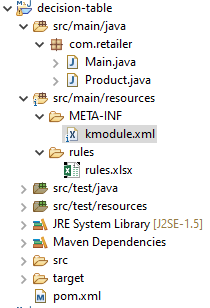
**Filter Cross Product** on Product where Department is “grocery”





**Decision Tables**: using Excel SpreadSheet

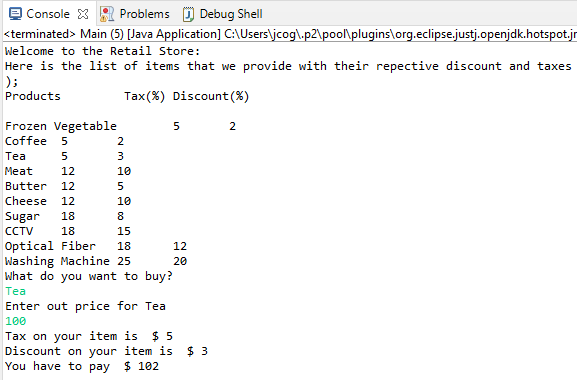




kmodule.xml



Output: Decision Table



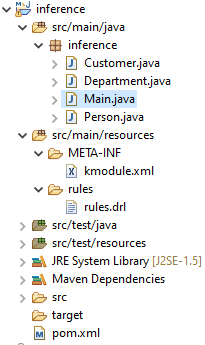
**Inference:**

Create a customer inference or the customer is determined by the name of the Department and the employee type being a student.

An inference runs in all locations without rewriting code for every single change in the when clause.

Both the department and student requirements are checked in the same when clause. Then insert the true condition into the pseudo-list “Customer”. If the person with the true conditions is the customer entering the department, then we execute the discount.

A change in department is a change in one location that has the same code to determine the discount.



Main.java



Customer.java



Department.java



Person.java



rules.drl



kmodule.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<kmodule xmlns=*"http://jboss.org/kie/6.0.0/kmodule"*>

<kbase name=*"rules"* packages=*"rules"*>

<ksession name=*"ksession-rule"*/>

</kbase>

</kmodule>

pom.xml

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>com.inference</groupId>

<artifactId>inference</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.kie</groupId>

<artifactId>kie-api</artifactId>

<version>7.48.0.Final</version>

</dependency>

<dependency>

<groupId>org.drools</groupId>

<artifactId>drools-core</artifactId>

<version>7.48.0.Final</version>

</dependency>

<dependency>

<groupId>org.drools</groupId>

<artifactId>drools-compiler</artifactId>

<version>7.48.0.Final</version>

</dependency>

<dependency>

<groupId>org.drools</groupId>

<artifactId>drools-decisiontables</artifactId>

<version>7.48.0.Final</version>

</dependency>

<dependency>

<groupId>org.kie</groupId>

<artifactId>kie-ci</artifactId>

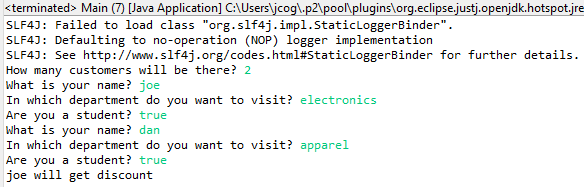
<version>7.48.0.Final</version>

</dependency>

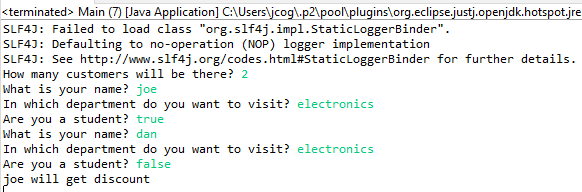
</dependencies>

</project>

Output: testing department

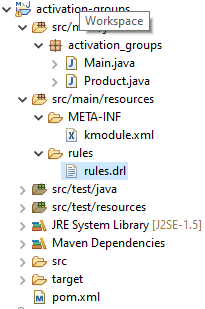


Output: testing student



**Activation Groups:**

Only one of rules that is part of the activation group can be fired.



Product.java



rules.drl



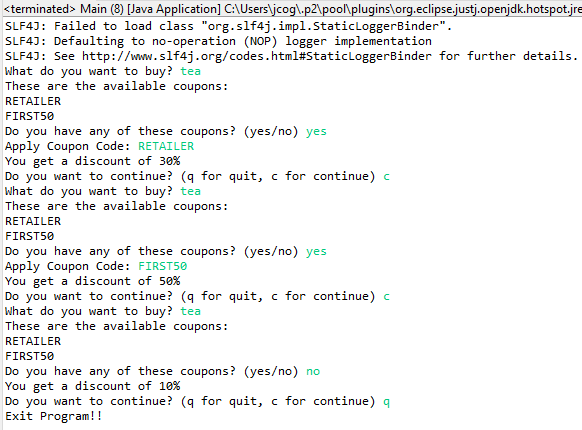
kmodules.xml

(See previous example)

pom.xml

(See previous example)

Output: test

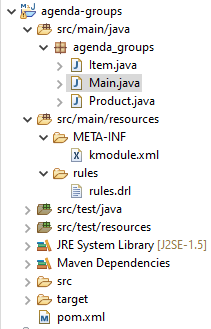


Main.java





**Agenda Groups** is a special way of Prioritizing Rules. You can override salience with Agenda Groups. This rule with the Agenda-Group will run first.



kmodules.xml

(See previous example)

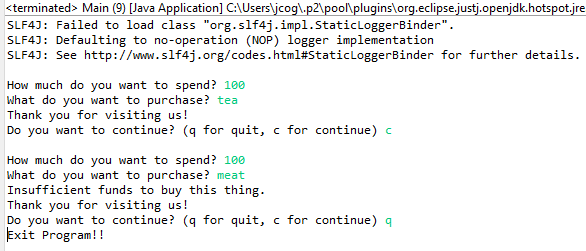
pom.xml

(See previous example)

rules.drl



Output: check



Main.java





Product.java

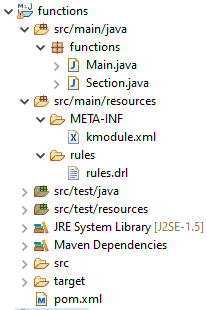


Item.java

**Functions:**

A piece of code that returns something typically a true or false.

Functions are called over and over again. A tax calculator function is an example.



rules.drl



kmodules.xml

(See previous example)

pom.xml

(See previous example)

Section.java



Main.java



Output:



SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

Here are the following sections in our building? apparel

electronics

grocery

cutlery

shoes

stationary

Enter a department: electronics

You will have to go 4 floor for shopping

Do you want to continue? (q for quit, c for continue) c

Here are the following sections in our building? apparel

electronics

grocery

cutlery

shoes

stationary

Enter a department: grocery

You will have to go 3 floor for shopping

Do you want to continue? (q for quit, c for continue) c

Here are the following sections in our building? apparel

electronics

grocery

cutlery

shoes

stationary

Enter a department: cutlery

You will have to go 5 floor for shopping

Do you want to continue? (q for quit, c for continue) c

Here are the following sections in our building? apparel

electronics

grocery

cutlery

shoes

stationary

Enter a department: shoes

You will have to go 2 floor for shopping

Do you want to continue? (q for quit, c for continue) c

Here are the following sections in our building? apparel

electronics

grocery

cutlery

shoes

stationary

Enter a department: stationary

You will have to go 6 floor for shopping

Do you want to continue? (q for quit, c for continue) q

Exit Program!!

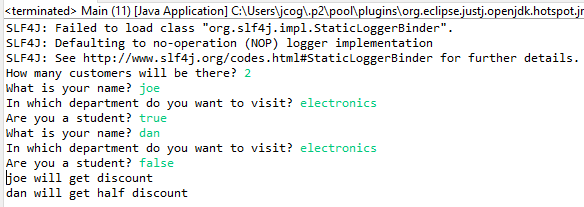
**InsertLogical:**

Truth maintenance is the result of insert logical pattern.

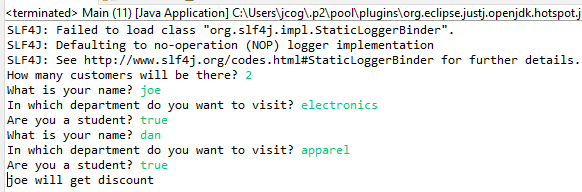
Truth maintenance is inference but with the ability to adapt to change.

InsertLogical updates changes whereas insert does not!

Output: testing for student



Output: testing for department

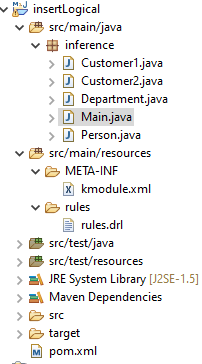


kmodules.xml

(See previous example)

pom.xml

(See previous example)



Customer1.java (list for full discount)



Customer2.java (list for half discount)



Department.java



Person.java



rules.drl



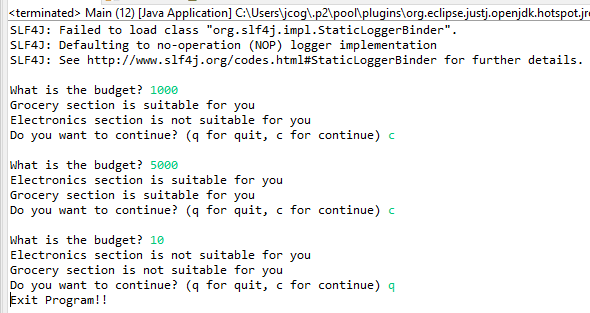
Main.java

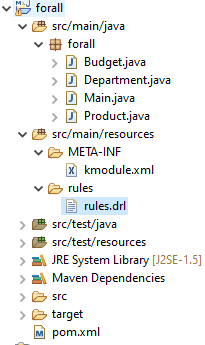


**ForAll:**

Loops

Output: testing with 3 limits





Budget.java

**package** forall;

**public** **class** Budget {

**private** **int** budget;

**public** **int** getBudget() {

**return** budget;

}

**public** **void** setBudget(**int** budget) {

**this**.budget = budget;

}

}

Department.java

**package** forall;

**public** **class** Department {

**private** String name;

**private** Product[] products;

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** Product[] getProducts() {

**return** products;

}

**public** **void** setProducts(Product[] products) {

**this**.products = products;

}

}

rules.drl



Product.java

**package** forall;

**public** **class** Product {

**private** String name;

Product[] products;

**private** **int** price;

**public** **int** getPrice() {

**return** price;

}

**public** **void** setPrice(**int** price) {

**this**.price = price;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** Product[] getProducts() {

**return** products;

}

**public** **void** setProducts(Product[] products) {

**this**.products = products;

}

}

Main.java

**package** forall;

**import** java.util.Scanner;

**import** org.kie.api.KieServices;

**import** org.kie.api.runtime.KieContainer;

**import** org.kie.api.runtime.KieSession;

**public** **class** Main {

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

System.***err***.close();

System.*setErr*(System.***out***);

Scanner input = **null**;

Scanner inputExit = **new** Scanner(System.***in***);

String quit = "c";

**try** {

KieServices ks = KieServices.Factory.*get*();

KieContainer kContainer = ks.getKieClasspathContainer();

KieSession kSession = kContainer.newKieSession("ksession-rule");

**while**(!quit.equals("q")) {

Department grocery = **new** Department();

grocery.setName("Grocery");

Product[] groceryProducts;

groceryProducts = **new** Product[4];

Product tea = **new** Product();

tea.setName("tea");

tea.setPrice(40);

groceryProducts[0] = tea;

Product coffee = **new** Product();

coffee.setName("coffee");

coffee.setPrice(50);

groceryProducts[1] = coffee;

Product chicken = **new** Product();

chicken.setName("chicken");

chicken.setPrice(100);

groceryProducts[2] = chicken;

Product meat = **new** Product();

meat.setName("meat");

meat.setPrice(200);

groceryProducts[3] = meat;

Department electronics = **new** Department();

electronics.setName("Electronics");

Product[] electronicsProducts;

electronicsProducts = **new** Product[4];

Product tv = **new** Product();

tv.setName("TV");

tv.setPrice(4000);

electronicsProducts[0] = tv;

Product laptop = **new** Product();

laptop.setName("LAPTOP");

laptop.setPrice(5000);

electronicsProducts[1] = laptop;

Product oven = **new** Product();

oven.setName("OVEN");

oven.setPrice(1500);

electronicsProducts[2] = oven;

Product pc = **new** Product();

pc.setName("PC");

pc.setPrice(5000);

electronicsProducts[3] = pc;

System.***out***.println();

System.***out***.print("What is the budget? ");

input = **new** Scanner(System.***in***);

**int** funds = input.nextInt();

Budget budget = **new** Budget();

budget.setBudget(funds);

electronics.setProducts(electronicsProducts);

grocery.setProducts(groceryProducts);

kSession.insert(electronics);

kSession.insert(grocery);

kSession.insert(budget);

kSession.fireAllRules();

System.***out***.print("Do you want to continue? (q for quit, c for continue) ");

quit = inputExit.next();

// reset session

kSession = kContainer.newKieSession("ksession-rule");

}

System.***out***.println("Exit Program!!");

} **catch**(Exception e) {

e.printStackTrace();

}

inputExit.close();

input.close();

}

}

kmodules.xml

(See previous example)

pom.xml

(See previous example)