Использование Decision Table в JBoss Drools

Правила. Мы знаем это слово с самого детства. В начале родители учат нас, как нужно правильно поступать, потом мы приходим в школу и там учителя диктуют свои порядки. В университете мы опять же сталкиваемся с правилами, которые для нас устанавливают преподаватели и деканат. По мере взросления мы начинаем уже следовать законам, которые разработаны государством. Что же общего у всех этих правил? То, что они выверены годами, десятилетиями, а некоторые даже и поколениями. Представьте какова была бы наша жизнь, если бы правила менялись часто: каждую неделю или каждый день!  
  
У нас в компании Mediascope в ежедневной поставке данных мы тоже используем правила: для расчета демографических атрибутов респондентов и домохозяйств, для расчета характеристик категорий товаров или брендов. Все эти правила меняются достаточно часто и каждый раз переделывать конвейер поставки данных было бы накладно. Поэтому мы решили вынести эти правила отдельно от кода и отдать их на поддержку бизнес-аналитикам. В данной статье я хотел бы показать как мы пользуемся этими правилами, а также продемонстрировать основные конструкции и практики использования этих правил.

*<!-------------------------------------------------------------------------------------------------------------------------------------->*

Что это вообще такое?

Довольно часто в процессе разработки коммерческого софта мы сталкиваемся с проблемой, когда некоторую логику расчета необходимо передать на сторону заказчика. Зачастую представитель этого заказчика не хочет (или не может) использовать какой-либо язык программирования для описания необходимой логики работы приложения. В таких случаях на помощь приходят BRMS или Business Rule Management System. Это такая информационная система для создания, управления и исполнения той самой  
бизнес-логики приложения (её еще называют бизнес-правилами). Обычно такие системы состоят из сервера, на котором происходит выполнение правил - это юрисдикция программистов, и средств ведения самих правил - это уже зона ответственности бизнеса.  
   
Практически каждая компания, где бизнес плотно взаимодействует с программной разработкой, пытается изобрести свой "велосипед". И мы не исключение. После первой итерации разработки системы управления правилами, было принято решение отказаться от внутреннего решения в пользу уже существующих. Причин было несколько: недостаточно быстрая работа системы, неудобное описание правил, постоянно возникающие ошибки в расчетах, и что самое страшное, непредсказуемо меняющийся выходной результат. Хорошо, что мы не внедрили это решение в продакшен!   
  
BRMS фреймворков на рынке достаточно много. Свои решения предлагают многие крупные компании: IBM, Red Hat, Agiloft, SAS и даже Bosch. Все они – либо платные, либо не подходили нам по тем или иным критериям. Мы решили начать с уже зарекомендовавшей себя системы JBoss Drools. Она достаточно надежная, проверенная временем, используется в банковских решениях, ритейле и телекоме, а также предоставляет возможность ведения бизнес-правил, как на специальном языке DRL, так и с помощью Excel-таблиц. Существует также и несколько UI решений для ведения правил. Так как наши аналитики используют единую систему ведения справочников и нам удобнее хранить правила там, поэтому от использования UI мы отказались в пользу понятных бизнесу Excel-таблиц.

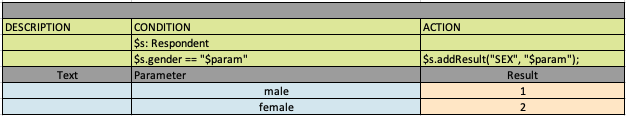
Что же такое бизнес-правило и как оно выглядит?  
  
Обычно под бизнес-правилом понимается набор инструкций или ограничений, которые позволяют принимать решение о выполнении того или иного действия. Другими словами, это некий набор логики, который позволяет в зависимости от вариации входных значений выполнять определенные действия, соответствующие входным параметрам. Давайте рассмотрим простой пример.  
  
У нас есть респондент, который обладает характеристикой *Гендерная идентичность (gender)*. В зависимости от значения этой характеристики респонденту выставляется *Пол (SEX).* То есть, если *gender = male*, то значению свойства пол нужно будет поставить *1*. В противном случае это будет *2*. На языке DRL это правило будет выглядеть следующим образом:

*rule "Rule 1 Example 1"  
 when  
 $s: Respondent($s.gender == "male")  
 then  
 $s.addResult("SEX", "1");  
end  
  
 rule "Rule 2 Example 2"  
 when  
 $s: Respondent($s.gender == "female")  
 then  
 $s.addResult("SEX", "2");  
 end*

Таким образом, мы имеем некий синтаксис очень похожий на язык Gherkin. В нашем случае у нас есть два правила, которые состоят из условия (указанного после ключевого слова **when**), и результата, которой будет получен в ходе выполнения этого условия (идет после слова **then)**. Условия в терминологии Drools принято называть Left Hand Side (или LHS), а действия - Right Hand Side (или RHS). Стоит упомянуть еще об одном объекте: *Respondent***.** Это **fact,** т.е. объект в текущей памяти Drools, над которым будут производиться те или иные преобразования. В нашем случае у этого объекта присутствуют свойства *gender*и *result*. Для того, чтобы было удобнее работать с объектами, framework предусматривает возможность введения переменных. Обычно переменные начинаются с символа **$**.

Возможно, вы подумали, что все это усложняет достаточно простую логику и можно обойтись обычной конструкцией *if – else if – else*. Но в случае большого правила и разных пересекающихся условий, использовать стандартные средства языка программирования уже становиться не удобно. В прочем, при большом количестве условий использовать DRL-синтаксис также не представляется возможным, к тому же не всегда представители бизнес-аналитики хотят углубляться в изучение такого описания. В таком случае на помощь нам приходят всеми горячо любимые (а особенно аналитиками) Excel-таблицы. Drools в этом случае предоставляет нам достаточно широкий функционал их использования.

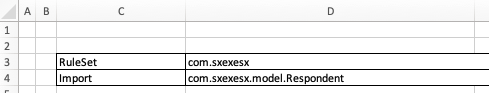
Вот так это правило будет выглядеть на языке Excel-таблиц:



Как же создать таблицу правил?

Прежде, чем мы перейдем к описанию основных конструкций, используемых в таблицах, необходимо разобраться, как же правильно формировать эту таблицу. Стоит отметить, что Drools умеет работать как с таблицами, созданными в табличных редакторах, Miscrosoft Excel или OpenOffice, так и с форматом CSV. Перед применением правил они будут сконвертированы из табличной формы в DRL формат.

В любой таблице правил можно выделить две области: область настройки правил и область описания правил. Первую обычно можно узнать по зарезервированному слову **RuleSet**, тогда как вторая – это **RuleTable**. Обратите внимание, что все зарезервированные чувствительны к регистру.



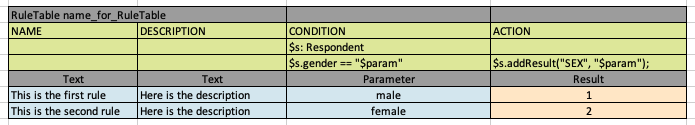
В области настройки правил обычно указываются основные конструкции DRL формата и атрибуты самих правил. Обычно это пара - зарезервированное слово в левой ячейке и значение в правой. Полный список зарезервированных слов можно найти в документации. Вот часть из них:

- **RuleSet** – здесь указывается имя пакета для сгенерированного файла DRL. Это параметр обязательно должен стоять первым.

- **Import** – через запятую указываются факты с указанием пакета, а также такие Java-классы, которые можно использовать в расчетах, например, java.lang.Math

- **Functions** – здесь можно описывать функции, которые будут работать в рамках данного RuleSet’а. Функции должны соответствовать DRL синтаксису.

В области описания правил тоже есть свои зарезервированные слова, при этом главным словом является **RuleTable**, которое указывает на то, что таблица ниже соответствует таблице правил, которую движок Drools должен преобразовать в DRL-синтаксис. Опционально можно указать название для таблицы правил. В нашем случае это **name\_for\_RuleTable**.



Начиная со следующей строки идут колонки:

**NAME** – имя правила. Его можно не указывать.

**DESCRIPTION** – расшифровка для правила. Его тоже можно не указывать. Эти два параметра нужны для того, чтобы не потеряться в большом количестве правил.

**CONDITION** – это то Left Hand Side или указание условия на основании которого будет выполнен ACTION. Этот параметр обязателен.

**ACTION** – действие, которое необходимо применить к факту. В нашем случае это метод addResult, который добавляет в Map, значения результирующих переменных. Этот параметр обязателен. По факту в блоке ACTION выполняется Java код, поэтому наличие точки с запятой здесь также обязательно. Также, через точку с запятой можно указать сколько угодно методов.

Следует отметить, что количество CONDITION и ACTION может быть больше одного.

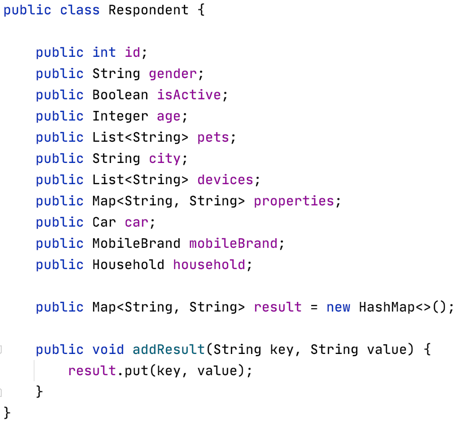
Как отмечалось ранее, строкой ниже идет присваивание переменной ***$s*** факта *Respondent*. Отмечу, что его полное имя с названием пакета в котором он находится, обязательно указать в параметре **Import**. Если у нас несколько CONDITION относятся к одному факту, тогда ячейки с фактом необходимо объединить в одну. Также мы можем работать с разными фактами в разных CONDITION колонках, для этого мы просто указываем новый факт и новую переменную, не забыв добавить его в **Import**.

Далее, на следующей строке, у нас идет описание самих правил в колонке CONDITION и действий в колонке ACTION, которые необходимо выполнить. О том, какие могут быть условия и действия к выполнению, будет описано ниже. А пока перейдем к следующей строке. Это строка заголовков полей (Text-Parameter-Result на картинке). Ее указывать обязательно. В противном случае, те переменные/условия, которые будут указаны вместо этой строки, Drools проигнорирует. Далее у нас идет указание самих условий.

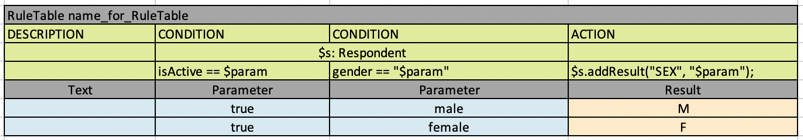
Какие условия могут быть?

В интернете достаточно много информации о том, как писать условия, но она достаточно разрозненная. Главной задачей этой статьи было собрать воедино все часто встречающиеся варианты.

Для начала стоит упомянуть еще раз, что все манипуляции мы будем производить над объектом Respondent. В нашей терминологии респондент – это лицо, которое принимает участие в исследовании. У каждого респондента есть свой набор свойств (например, гендерная принадлежность, рассмотренная ранее). Для того, чтобы показать все многообразие условий, с которыми работает Drools, я вынес каждое свойство в отдельное поле класса *Respondent*. Для простоты понимания, в качестве еще одного поля класса добавляем *Map<String, String> result*, в котором будем собирать все результирующие переменные. Таким образом, класс *Respondent* будет выглядеть следующим образом:



Самое простое правило по определению гендерной принадлежности мы уже рассмотрели. А что, если нам нужно определить половой признак, только у тех респондентов, которые участвуют в исследовании? Для фильтрации таких респондентов мы будет использовать булеву переменную *isActive.* Значение *true* – респондент участвует в исследовании, *false –* нет.



На рисунке выше показано, как мы объединили 2 условия: активность респондента и его половой признак. Не трудно догадаться, что в таком случае происходит объединение правил, находящихся в одной строке, по логическому *И*. Два условия объединены одним фактом *Respondent*. Вот так выглядит описание правила на языке DRL:

*rule "name\_for\_RuleTable\_20"*

*when*

*$s: Respondent(isActive == true, gender == "male")*

*then*

*$s.addResult("SEX", "M");*

*end*

*rule "name\_for\_RuleTable\_21"*

*when*

*$s: Respondent(isActive == true, gender == "female")*

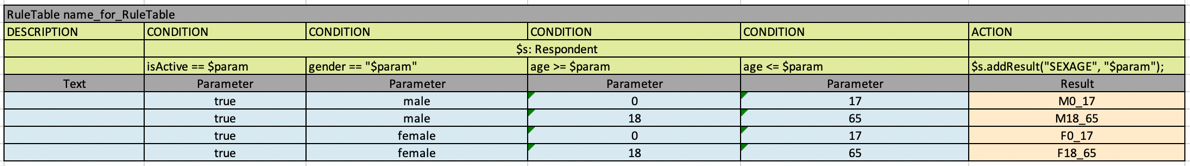
*then*

*$s.addResult("SEX", "F");*

*end*

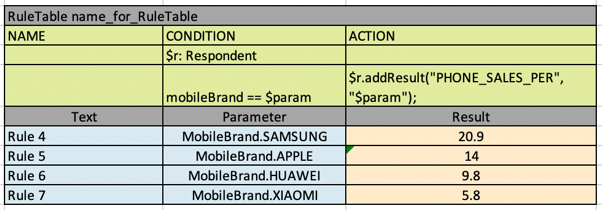
Стоит отметить использование параметра *$param -* этот параметр работает в рамках одного столбца и во время компиляции правил он будет заменен на конкретное значение из ячейки. Т.е. условие *isActive == $param* будет преобразовано в *isActive == true.* В случае работы с булевыми или целочисленными переменными экранировать его не надо. Движок Drools понимает, что это не строка. В случае работы со строками экранирование обязательно, как показано в примере со свойством gender.

В социологических исследования часто разбивают респондентов на половозрастные группы. Следующий пример как раз демонстрирует такое разбиение:

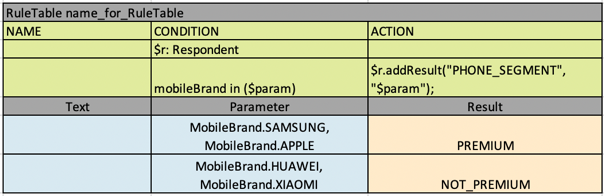


Данное правило производит разбивку активных респондентов по половому признаку, а также на две возрастные категории до 17 лет включительно и после 18 лет. Повторюсь, в случае работы с числами экранирующие кавычки не требуются.

Еще один пример, где кавычки не требуются – это перечисления. Допустим, каждый из респондентов обладает мобильным телефоном. В зависимости от того, какого бренда телефон, мы запишем в результирующую переменную *PHONE\_SALES\_PER* значение процента продаж на мировом рынке мобильных устройств за 2019 год. Так как результирующий словарь в качестве значения принимает строку, то *$param* необходимо экранировать. Также стоит отметить, что *MobileBrand* необходимо добавить в настройку **Import** в области описания правил.

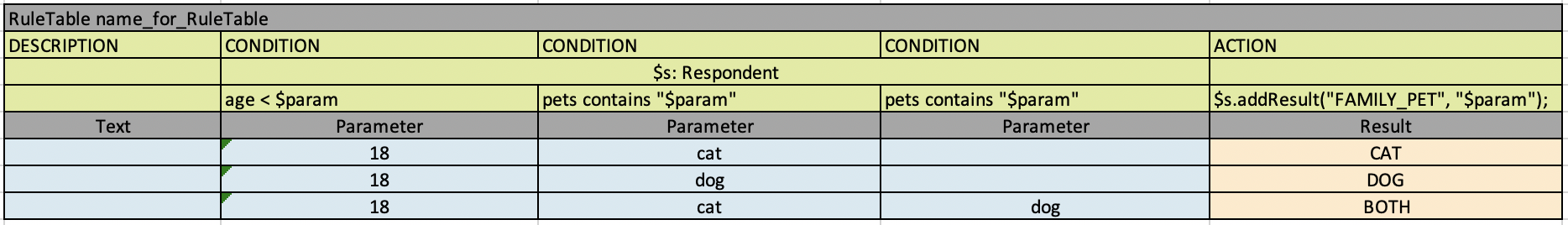


Иногда встречается такие условия, когда одинаковое действие нужно применить к разным условиям. Такого поведения можно добиться, если в колонке ACTION дублировать значения ***$param***, то есть создавать отдельные правила с разными условиями и одинаковыми результатами. Но это будет загромождать нашу таблицу. Для таких случаев предусмотрена специальная конструкция ***in:***

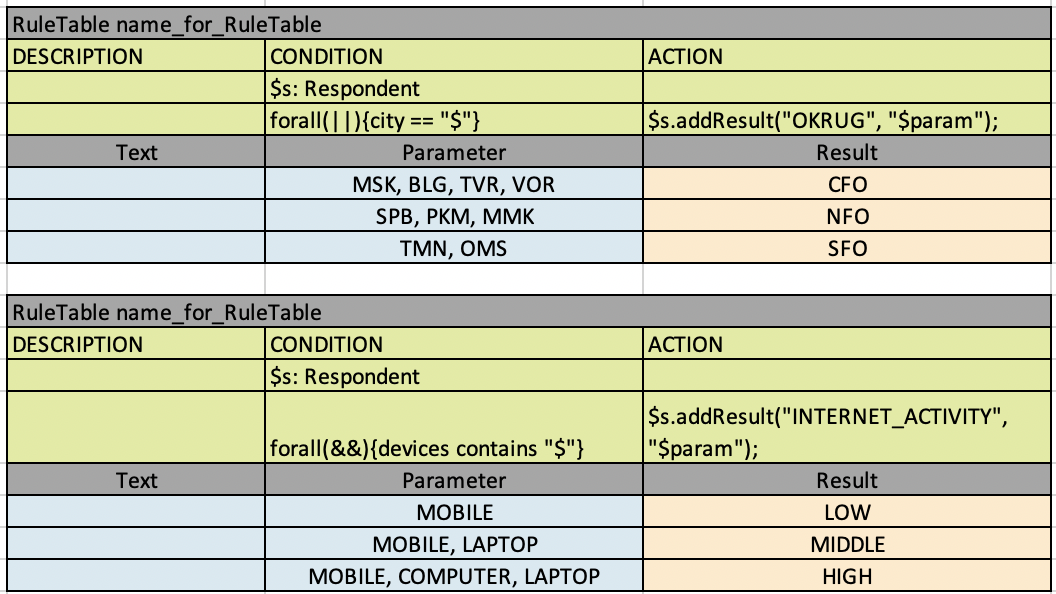


В данном примере мы будем иметь два правила: если бренд мобильного телефона нашего респондента марки SAMSUNG или APPLE, тогда в результирующую переменную мы запишем значение PREMIUM. В случае, когда бренд телефона HUAWEI или XIAOMI – значение будет NOT\_PREMIUM. (Прошу меня извинить владельцев данных смартфонов, что отнес их к [непривилегированным](https://www.google.com/search?client=safari&rls=en&q=%D0%BD%D0%B5%D0%BF%D1%80%D0%B8%D0%B2%D0%B8%D0%BB%D0%B5%D0%B3%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%BC&spell=1&sa=X&ved=2ahUKEwiztpu4vpjtAhXyk4sKHa_0Dw8QkeECKAB6BAgJEC8) :) Другими словами, мы выбираем значение из массива значений.

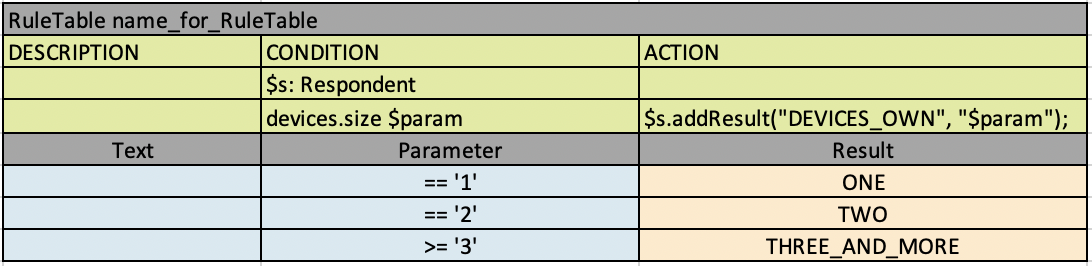
А если у нас обратная задача, когда мы имеем массив значений и при вхождении конкретного значения, выполняется действие? В подобном случае мы используем конструкцию ***contains***. Эта конструкция работает со всеми объектами интерфейса *java.util.Collection*. В нашем примере у объекта *Respondent* есть коллекция домашних животных *pets.* Мы сделаем выборку тех респондентов, которые не достигли совершеннолетия и указали в своих ответах в качестве домашнего животного кошку или собаку. Третье правило сработает в том случае, если в семье есть и собака и кошка. Т.к. домашнее животное – это строковая переменная, то переменную *$param* мы заключаем в кавычки.



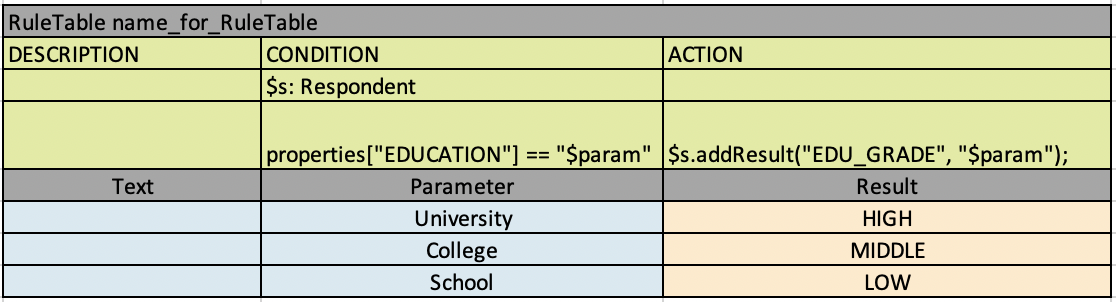
В следующих двух примерах мы рассмотрим еще один оператор работы с коллекциями: ***forall(<оператор>){<условие>}.*** В первом примере правило сработает в случае выполнения условия в фигурных скобках и хотя бы одного из перечисленных через запятую параметров в ячейке. Во втором примере, должно выполниться условие в фигурных скобках со всеми параметрами, перечисленными через запятую в соответствующей ячейке. Другими словами, в первом примере объединение условий происходит через логическое *ИЛИ*, тогда как во втором через *И*. Собственно это условие и указывается в фигурных скобках. Важно отметить, что переменная, которая будет обращаться к значению ячейки, в данном случае указывается без слова ***param***. Еще один интересный момент, который я хотел бы отметить в примерах ниже – это отсутствие ссылки на факт (***$s***). Дело в том, что если мы работаем с одним фактом, то Drools понимает, что мы обращаемся полям класса-факта (*city* и *devices*), поэтому указание переменной в описании CONDITION можно опустить.

****

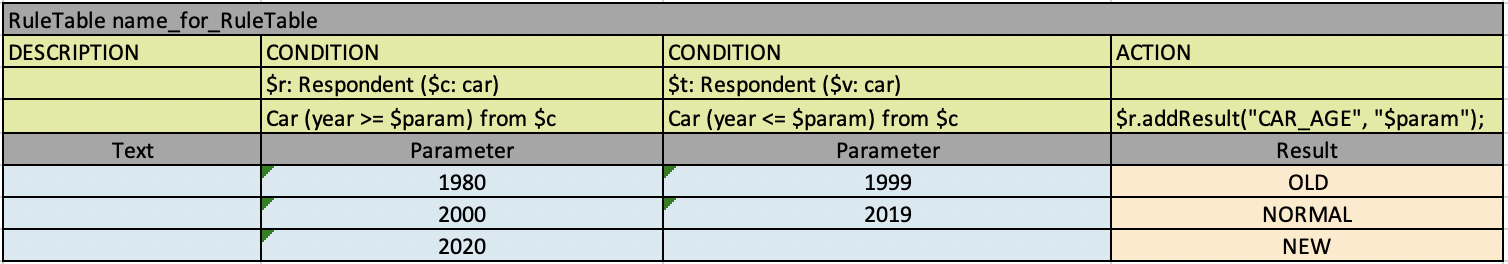
Последнее, что хотелось бы рассмотреть при работе с коллекциями – это количество значений. Обращение к количеству элементов происходит через функцию ***size****:*

****

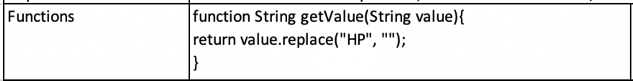
Когда у объекта-факта много вложенных свойств это немного затрудняет понимание объекта, а также работу с ним. Поэтому мы обычно стараемся все свойства объекта привести к словарю с парой ключ-значение. Drools имеет удобный механизм обращения к значениям словаря по ключу:



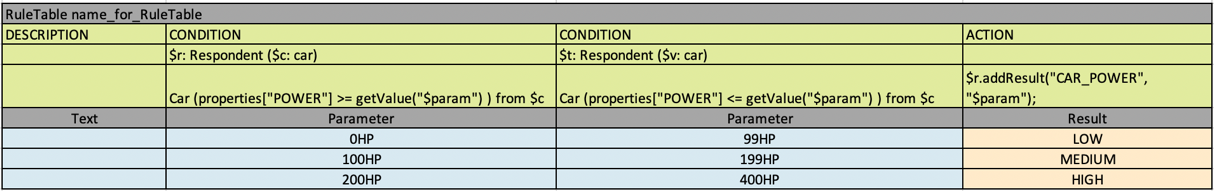
В нашей практике мы достаточно часто получаем значение с древовидными структурами, когда объект имеет вложенные классы со своими наборами свойств. Предлагаю разобрать несколько таких примеров. Допустим у каждого респондента есть вложенный объект *Car* c набором свойств: порядковый номер, марка автомобиля, модель и год выпуска. И, к примеру, нам необходимо категоризировать автомобиль по году выпуска. Для того, чтобы обратиться к объекту в родительском объекте-факте необходимо явно указать новую переменную и вложенный объект. В ячейке описания условия нужно сослаться на этот дочерний объект с помощью ключевого слова ***from*:**



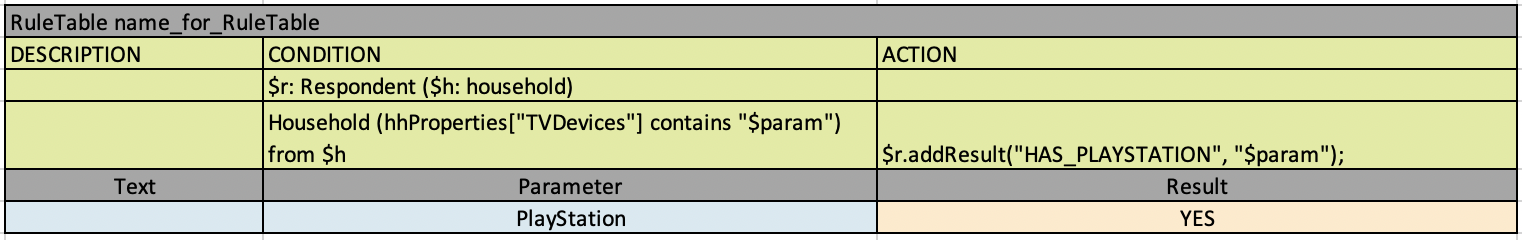
Достаточно удобно объединять вложенные объекты и их наборы свойств, которые представлены в виде словаря. В примере описанном ниже, у объекта *Car,* который является частью факта *Respondent* есть *Map<String, String> properties* с набором свойств. В данном наборе правил мы обращаемся к свойству *POWER*. Также в описании условий правил присутствует функция Drools ***getValues.*** Ее реализацию необходимо вынести в блок ***Functions*** в области настройки правил.



И само правило:



В заключении описания возможных реализаций условий, хотелось бы показать комплексный пример использования словаря во вложенном объекте, где в качестве значения по ключу используется массив данных. Условие задачи звучит следующим образом: необходимо отобрать тех респондентов, у которых в домохозяйстве есть приставка PlayStation. Другими словами, у дочернего объекта *Household* факта *Respondent* необходимо найти свойство *TVDevices* и проверить наличие приставки PlayStation. Правило будет выглядеть следующим образом:



Как все это применять?

Все описанные выше условия мы можем комбинировать различным образом между собой или создавая длинные цепочки правил через логическое объединение. Но как мы можем эффективно использовать блок ACTION? Как я уже отмечал, в этом блоке мы можем воспользоваться всем арсеналом возможностей Java. В описанных примерах я использовал достаточно тривиальный метод добавления значения в словарь. Очень часто (особенно для отладки правил) удобно использовать в ACTION логирование или вывод на консоль.

Но есть и ряд зарезервированных выражений Drools, которые позволяют сделать расчет правил еще гибче:

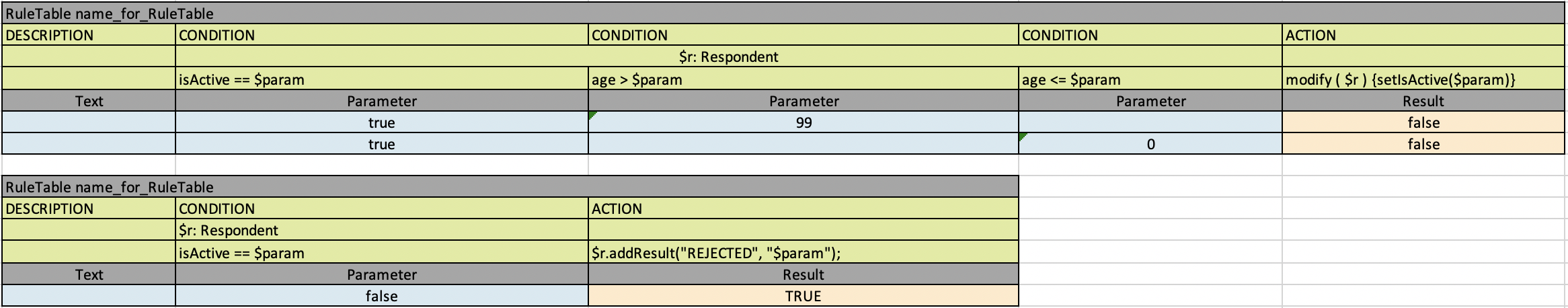
- ***set –*** устанавливает значение свойства факта, но не уведомляет движок Drools об этом.

***- update –*** уведомляет движок о том, что факт был изменен посредством одного или нескольких ***set***

***- modify –*** это некая сумма двух предыдущих действий: здесь происходит установка одного или нескольких свойств факта, а также уведомляется движок об изменении факта

***- insert –*** добавление нового факта

Приведу пример наиболее часто используемого выражения ***modify***.

******

В первом *RuleTable* мы производим отбор тех активных респондентов, у которых по каким-то причинам возраст больше 99 лет и меньше 0. Если такие респонденты найдены, то выражение изменит факт ***$r***, т.е. Respondent’a и установит ему значение *false.* В следующем *RuleTable* мы уже будем иметь измененное свойство *isActive*, и неактивным респондентам запишется свойству REJECTED значение TRUE.

В заключение…

Моей задачей было рассказать об основном функционале использования логики для описания CONDITION, безусловно, есть и другие варианты использования. Все приведенные мной примеры упрощены для того, чтобы было легче начать использование разных конструкций, как опытным программистам, так и начинающим. В реальном продакшене правила получаются гораздо сложнее и объемнее. Drools позволяет реализовать достаточно сложную логику. В официальной документации есть даже примеры реализации игр, но, к сожалению, без помощи табличного представления.

Информацию по запуску Drools и работе с ним работать можно с легкостью найти в интернете или форкайте мой проект с примерами на гитлаб ([*https://github.com/sxexesx/drools-decision-table*](https://github.com/sxexesx/drools-decision-table)).

Спасибо за внимание! Надеюсь данная статья окажется полезна тем, кто хочет погрузиться в удивительный мир BRMS!