## Pet Store Example

**Name:** Pet Store

**Main class:** org.drools.examples.petstore.PetStoreExample

**Module:** drools-examples

**Type:** Java application

**Rules file:** PetStore.drl

**Objective:** Demonstrate use of Agenda Groups, Global Variables and integration with a GUI,

including callbacks from within the rules

The Pet Store example shows how to integrate Rules with a GUI, in this case a Swing based desktop application. Within the rules file, it demonstrates how to use Agenda groups and auto-focus to control which of a set of rules is allowed to fire at any given time. It also illustrates the mixing of the Java and MVEL dialects within the rules, the use of accumulate functions and the way of calling Java functions from within the ruleset.

All of the Java code is contained in one file, PetStore.java, defining the following principal classes (in addition to several classes to handle Swing Events):

* Petstore contains the main() method that we will look at shortly.
* PetStoreUI is responsible for creating and displaying the Swing based GUI. It contains several smaller classes, mainly for responding to various GUI events such as mouse button clicks.
* TableModel holds the table data. Think of it as a JavaBean that extends the Swing class AbstractTableModel.
* CheckoutCallback allows the GUI to interact with the Rules.
* Ordershow keeps the items that we wish to buy.
* Purchase stores details of the order and the products we are buying.
* Product is a JavaBean holding details of the product available for purchase, and its price.

Much of the Java code is either plain JavaBeans or Swing-based. Only a few Swing-related points will be discussed in this section, but a good tutorial about Swing components can be found at Sun's Swing website, in [http://java.sun.com/docs/books/tutorial/uiswing/](http://docs.jboss.org/drools/release/5.4.0.Beta1/drools-expert-docs/html_single/index.html???) .

The pieces of Java code in Petstore.java that relate to rules and facts are shown below.

**Example 9.52. Creating the PetStore RuleBase in PetStore.main**

KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();

kbuilder.add( ResourceFactory.newClassPathResource( "PetStore.drl",

                                                    PetStore.class ),

              ResourceType.DRL );

KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();

kbase.addKnowledgePackages( kbuilder.getKnowledgePackages() );

// Create the stock.

Vector<Product> stock = new Vector<Product>();

stock.add( new Product( "Gold Fish", 5 ) );

stock.add( new Product( "Fish Tank", 25 ) );

stock.add( new Product( "Fish Food", 2 ) );

// A callback is responsible for populating the

// Working Memory and for firing all rules.

PetStoreUI ui = new PetStoreUI( stock,

                                new CheckoutCallback( kbase ) );

ui.createAndShowGUI();

The code shown above loads the rules from a DRL file on the classpath. Unlike other examples where the facts are asserted and fired straight away, this example defers this step to later. The way it does this is via the second last line where a PetStoreUI object is created using a constructor accepting the Vector object stock collecting our products, and an instance of the CheckoutCallback class containing the Rule Base that we have just loaded.

The Java code that fires the rules is within the CheckoutCallBack.checkout() method. This is triggered (eventually) when the Checkout button is pressed by the user.

**Example 9.53. Firing the Rules - extract from CheckoutCallBack.checkout()**

public String checkout(JFrame frame, List<Product> items) {

    Order order = new Order();

    // Iterate through list and add to cart

    for ( Product p: items ) {

        order.addItem( new Purchase( order, p ) );

    }

    // Add the JFrame to the ApplicationData to allow for user interaction

    StatefulKnowledgeSession ksession = kbase.newStatefulKnowledgeSession();

    ksession.setGlobal( "frame", frame );

    ksession.setGlobal( "textArea", this.output );

    ksession.insert( new Product( "Gold Fish", 5 ) );

    ksession.insert( new Product( "Fish Tank", 25 ) );

    ksession.insert( new Product( "Fish Food", 2 ) );

    ksession.insert( new Product( "Fish Food Sample", 0 ) );

    ksession.insert( order );

    ksession.fireAllRules();

    // Return the state of the cart

    return order.toString();

}

Two items get passed into this method. One is the handle to the JFrame Swing component surrounding the output text frame, at the bottom of the GUI. The second is a list of order items; this comes from the TableModel storing the information from the "Table" area at the top right section of the GUI.

The for loop transforms the list of order items coming from the GUI into the Order JavaBean, also contained in the file PetStore.java. Note that it would be possible to refer to the Swing dataset directly within the rules, but it is better coding practice to do it this way, using simple Java objects. It means that we are not tied to Swing if we wanted to transform the sample into a Web application.

It is important to note that all state in this example is stored in the Swing components, and that the rules are effectively stateless. Each time the "Checkout" button is pressed, this code copies the contents of the Swing TableModel into the Session's Working Memory.

Within this code, there are nine calls to the Working Memory. The first of these creates a new Working Memory, as a Stateful Knowledge Session from the Knowledge Base. Remember that we passed in this Knowledge Base when we created the CheckoutCallBack class in the main() method. The next two calls pass in two objects that we will hold as global variables in the rules: the Swing text area and the Swing frame used for writing messages.

More inserts put information on products into the Working Memory, as well as the order list. The final call is the standard fireAllRules(). Next, we look at what this method causes to happen within the rules file.

**Example 9.54. Package, Imports, Globals and Dialect: extract from PetStore.drl**

package org.drools.examples

import org.drools.WorkingMemory

import org.drools.examples.petstore.PetStoreExample.Order

import org.drools.examples.petstore.PetStoreExample.Purchase

import org.drools.examples.petstore.PetStoreExample.Product

import java.util.ArrayList

import javax.swing.JOptionPane;

import javax.swing.JFrame

global JFrame frame

global javax.swing.JTextArea textArea

The first part of file PetStore.drl contains the standard package and import statements to make various Java classes available to the rules. New to us are the two globals frame and textArea. They hold references to the Swing components JFrame and JTextArea components that were previously passed on by the Java code calling the setGlobal() method. Unlike variables in rules, which expire as soon as the rule has fired, global variables retain their value for the lifetime of the Session.

The next extract from the file PetStore.drl contains two functions that are referenced by the rules that we will look at shortly.

**Example 9.55. Java Functions in the Rules: extract from PetStore.drl**

function void doCheckout(JFrame frame, WorkingMemory workingMemory) {

    Object[] options = {"Yes",

                        "No"};

    int n = JOptionPane.showOptionDialog(frame,

        "Would you like to checkout?",

        "",

        JOptionPane.YES\_NO\_OPTION,

        JOptionPane.QUESTION\_MESSAGE,

        null,

        options,

        options[0]);

    if (n == 0) {

        workingMemory.setFocus( "checkout" );

    }

}

function boolean requireTank(JFrame frame, WorkingMemory workingMemory, Order order, Product fishTank, int total) {

    Object[] options = {"Yes",

                        "No"};

    int n = JOptionPane.showOptionDialog(frame,

        "Would you like to buy a tank for your " + total + " fish?",

        "Purchase Suggestion",

        JOptionPane.YES\_NO\_OPTION,

        JOptionPane.QUESTION\_MESSAGE,

        null,

        options,

        options[0]);

    System.out.print( "SUGGESTION: Would you like to buy a tank for your "

                      + total + " fish? - " );

    if (n == 0) {

        Purchase purchase = new Purchase( order, fishTank );

        workingMemory.insert( purchase );

        order.addItem( purchase );

        System.out.println( "Yes" );

    } else {

        System.out.println( "No" );

    }

    return true;

}

Having these functions in the rules file just makes the Pet Store example more compact. In real life you probably have the functions in a file of their own, within the same rules package, or as a static method on a standard Java class, and import them, using import function my.package.Foo.hello.

The purpose of these two functions is:

* doCheckout() displays a dialog asking users whether they wish to checkout. If they do, focus is set to the checkOut agenda-group, allowing rules in that group to (potentially) fire.
* requireTank() displays a dialog asking users whether they wish to buy a tank. If so, a new fish tank Product is added to the order list in Working Memory.

We'll see the rules that call these functions later on. The next set of examples are from the Pet Store rules themselves. The first extract is the one that happens to fire first, partly because it has the auto-focus attribute set to true.

**Example 9.56. Putting items into working memory: extract from PetStore.drl**

// Insert each item in the shopping cart into the Working Memory

// Insert each item in the shopping cart into the Working Memory

rule "Explode Cart"

agenda-group "init"

auto-focus true

salience 10

dialect "java"

when

$order : Order( grossTotal == -1 )

$item : Purchase() from $order.items

then

insert( $item );

kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "show items" ).setFocus();

kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "evaluate" ).setFocus();

end

This rule matches against all orders that do not yet have their grossTotal calculated . It loops for each purchase item in that order. Some parts of the "Explode Cart" rule should be familiar: the rule name, the salience (suggesting the order for the rules being fired) and the dialect set to "java". There are three new features:

* agenda-group "init" defines the name of the agenda group. In this case, there is only one rule in the group. However, neither the Java code nor a rule consequence sets the focus to this group, and therefore it relies on the next attribute for its chance to fire.
* auto-focus true ensures that this rule, while being the only rule in the agenda group, gets a chance to fire when fireAllRules() is called from the Java code.
* kcontext....setFocus() sets the focus to the "show items" and "evaluate" agenda groups in turn, permitting their rules to fire. In practice, we loop through all items on the order, inserting them into memory, then firing the other rules after each insert.

The next two listings show the rules within the "show items" and evaluate agenda groups. We look at them in the order that they are called.

**Example 9.57. Show Items in the GUI - extract from PetStore.drl**

rule "Show Items"

agenda-group "show items"

dialect "mvel"

when

$order : Order( )

$p : Purchase( order == $order )

then

textArea.append( $p.product + "\n");

end

The "show items" agenda-group has only one rule, called "Show Items" (note the difference in case). For each purchase on the order currently in the Working Memory (or Session), it logs details to the text area at the bottom of the GUI. The textArea variable used to do this is one of the global variables we looked at earlier.

The evaluate Agenda group also gains focus from the "Explode Cart" rule listed previously. This Agenda group has two rules, "Free Fish Food Sample" and "Suggest Tank", shown below.

**Example 9.58. Evaluate Agenda Group: extract from PetStore.drl**

// Free Fish Food sample when we buy a Gold Fish if we haven't already bought

// Fish Food and don't already have a Fish Food Sample

rule "Free Fish Food Sample"

agenda-group "evaluate"

dialect "mvel"

when

$order : Order()

not ( $p : Product( name == "Fish Food") &amp;&amp; Purchase( product == $p ) )

not ( $p : Product( name == "Fish Food Sample") &amp;&amp; Purchase( product == $p ) )

exists ( $p : Product( name == "Gold Fish") &amp;&amp; Purchase( product == $p ) )

$fishFoodSample : Product( name == "Fish Food Sample" );

then

System.out.println( "Adding free Fish Food Sample to cart" );

purchase = new Purchase($order, $fishFoodSample);

insert( purchase );

$order.addItem( purchase );

end

// Suggest a tank if we have bought more than 5 gold fish and don't already have one

rule "Suggest Tank"

agenda-group "evaluate"

dialect "java"

when

$order : Order()

not ( $p : Product( name == "Fish Tank") &amp;&amp; Purchase( product == $p ) )

ArrayList( $total : size &gt; 5 ) from collect( Purchase( product.name == "Gold Fish" ) )

$fishTank : Product( name == "Fish Tank" )

then

requireTank(frame, drools.getWorkingMemory(), $order, $fishTank, $total);

end

The rule "Free Fish Food Sample" will only fire if

* we don't already have any fish food, and
* we don't already have a free fish food sample, and
* we do have a Gold Fish in our order.

If the rule does fire, it creates a new product (Fish Food Sample), and adds it to the order in Working Memory.

The rule "Suggest Tank" will only fire if

* we don't already have a Fish Tank in our order, and
* we do have more than 5 Gold Fish Products in our order.

If the rule does fire, it calls the requireTank() function that we looked at earlier (showing a Dialog to the user, and adding a Tank to the order / working memory if confirmed). When calling the requireTank() function the rule passes the global frame variable so that the function has a handle to the Swing GUI.

The next rule we look at is "do checkout".

**Example 9.59. Doing the Checkout - extract (6) from PetStore.drl**

rule "do checkout"

dialect "java"

when

then

doCheckout(frame, drools.getWorkingMemory());

end

The rule "do checkout" has **no agenda group set and no auto-focus attribute**. As such, is is deemed part of the default (MAIN) agenda group. This group gets focus by default when all the rules in agenda-groups that explicity had focus set to them have run their course.

There is no LHS to the rule, so the RHS will always call the doCheckout() function. When calling the doCheckout() function, the rule passes the global frame variable to give the function a handle to the Swing GUI. As we saw earlier, the doCheckout() function shows a confirmation dialog to the user. If confirmed, the function sets the focus to the checkout agenda-group, allowing the next lot of rules to fire.

**Example 9.60. Checkout Rules: extract from PetStore.drl**

rule "Gross Total"

agenda-group "checkout"

dialect "mvel"

when

$order : Order( grossTotal == -1)

Number( total : doubleValue )

from accumulate( Purchase( $price : product.price ), sum( $price ) )

then

modify( $order ) { grossTotal = total };

textArea.append( "\ngross total=" + total + "\n" );

end

rule "Apply 5% Discount"

agenda-group "checkout"

dialect "mvel"

when

$order : Order( grossTotal &gt;= 10 &amp;&amp; &lt; 20 )

then

$order.discountedTotal = $order.grossTotal \* 0.95;

textArea.append( "discountedTotal total=" + $order.discountedTotal + "\n" );

end

rule "Apply 10% Discount"

agenda-group "checkout"

dialect "mvel"

when

$order : Order( grossTotal &gt;= 20 )

then

$order.discountedTotal = $order.grossTotal \* 0.90;

textArea.append( "discountedTotal total=" + $order.discountedTotal + "\n" );

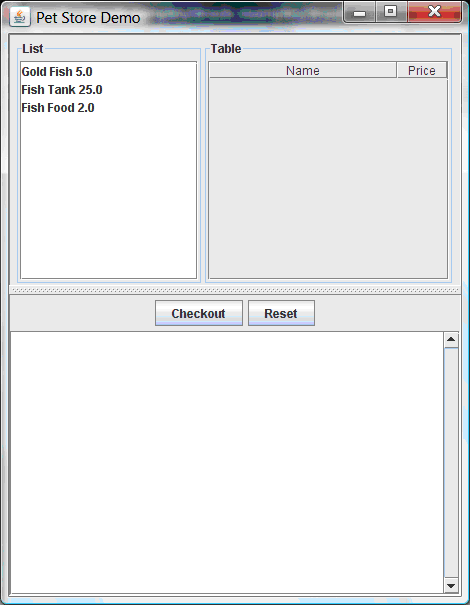
end

There are three rules in the checkout agenda-group:

* If we haven't already calculated the gross total, Gross Total accumulates the product prices into a total, puts this total into Working Memory, and displays it via the Swing JTextArea, using the textArea global variable yet again.
* If our gross total is between 10 and 20, "Apply 5% Discount" calculates the discounted total and adds it to the Working Memory and displays it in the text area.
* If our gross total is not less than 20, "Apply 10% Discount" calculates the discounted total and adds it to the Working Memory and displays it in the text area.

Now that we've run through what happens in the code, let's have a look at what happens when we actually run the code. The file PetStore.java contains a main() method, so that it can be run as a standard Java application, either from the command line or via the IDE. This assumes you have your classpath set correctly. (See the start of the examples section for more information.)

The first screen that we see is the Pet Store Demo. It has a list of available products (top left), an empty list of selected products (top right), checkout and reset buttons (middle) and an empty system messages area (bottom).

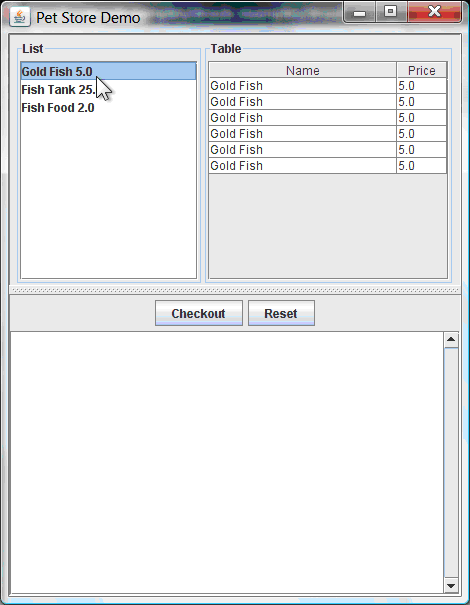


**Figure 9.13. PetStore Demo just after Launch**

To get to this point, the following things have happened:

1. The main() method has run and loaded the Rule Base but not yet fired the rules. So far, this is the only code in connection with rules that has been run.
2. A new PetStoreUI object has been created and given a handle to the Rule Base, for later use.
3. Various Swing components do their stuff, and the above screen is shown and waits for user input.

Clicking on various products from the list might give you a screen similar to the one below.

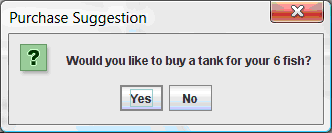


**Figure 9.14. PetStore Demo with Products Selected**

Note that no rules code has been fired here. This is only Swing code, listening for mouse click events, and adding some selected product to the TableModel object for display in the top right hand section. (As an aside, note that this is a classic use of the Model View Controller design pattern).

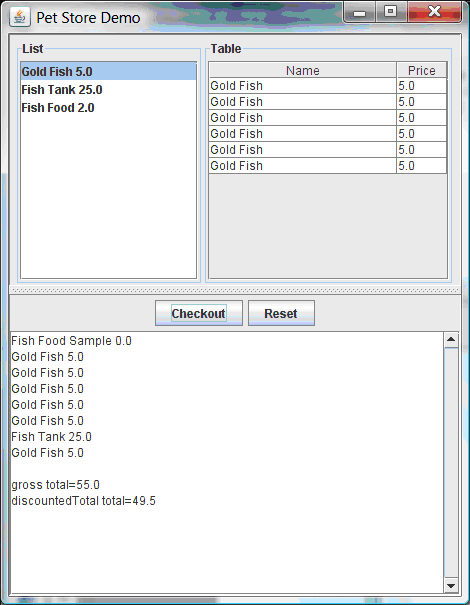
It is only when we press the "Checkout" button that we fire our business rules, in roughly the same order that we walked through the code earlier.

1. Method CheckOutCallBack.checkout() is called (eventually) by the Swing class waiting for the click on the "Checkout" button. This inserts the data from the TableModel object (top right hand side of the GUI), and inserts it into the Session's Working Memory. It then fires the rules.
2. The "Explode Cart" rule is the first to fire, given that it has auto-focus set to true. It loops through all the products in the cart, ensures that the products are in the Working Memory, and then gives the "Show Items" and Evaluation agenda groups a chance to fire. The rules in these groups add the contents of the cart to the text area (at the bottom of the window), decide whether or not to give us free fish food, and to ask us whether we want to buy a fish tank. This is shown in the figure below.



**Figure 9.15. Do we want to buy a fish tank?**

1. The Do Checkout rule is the next to fire as it (a) No other agenda group currently has focus and (b) it is part of the default (MAIN) agenda group. It always calls the doCheckout() function which displays a 'Would you like to Checkout?' Dialog Box.
2. The doCheckout() function sets the focus to the checkout agenda-group, giving the rules in that group the option to fire.
3. The rules in the the checkout agenda-group display the contents of the cart and apply the appropriate discount.
4. Swing then waits for user input to either checkout more products (and to cause the rules to fire again), or to close the GUI - see the figure below.



**Figure 9.16. Petstore Demo after all rules have fired.**

We could add more System.out calls to demonstrate this flow of events. The output, as it currently appears in the Console window, is given in the listing below.

**Example 9.61. Console (System.out) from running the PetStore GUI**

Adding free Fish Food Sample to cart

SUGGESTION: Would you like to buy a tank for your 6 fish? - Yes