JBDS & BRMS

Hands on Lab

BUSINESS RULES MANAGEMENT SERVER AND JBOSS DEVELOPER STUDIO

Table of Contents

Introduction 4

Overview 4

System Expectations 4

What is Expected of You 4

Lab Number 1: Installation of BRMS 5

Get the File 5

Lab Number 2: Installation of JBDS 6

Get the File 6

Running the Installer 6

Lab Number 3: Start the BRMS 16

Config File Change 16

Start BRMS 16

Lab Number 4: Explore the Sample Application 19

Explore the Sample Application 19

Lab Number 5: QA and Testing 24

Lab Number 6: JBDS Examples 28

Lab Number 6: Custom Rules with Complex Event Processing 37

Introduction

Overview

JBoss BRMS is a tool to help author, create, manage and test rules intended for business analysts. JBoss Developer Studio (JBDS) is intended for use by your technical developers in an organization. This lab expects that you have some knowledge around a rules engine, and will explore some of the features with how to get started using the BRMS to author, test, package, and publish rules. How to use JBDS to do the same, and then exercise those rules via soapUI ie as a Web Service This lab will walk you through setup, running the included examples, and authoring some new rules, test cases, and more via both JBoss Developer Studio and BRMS. With that said, most importantly have fun, and raise your hand with any questions.

Included FIles

Several files are included with this workshop. These files are all included in the Downloads directory on the Desktop in the BRMS and JBDS folder. You will need to get files out of the Downloads directory, and we will walk you through those steps.

System Expectations

It is expected that you have a Windows, Linux or Mac notebook and you are comfortable working and running Java programs on it. It is expected you will have the environment PATH set to include a JDK 6.0 to use for these labs. It is also a good idea to have JAVA_HOME set to your JDK that you plan on using. Please make sure you do this before running any of the labs. Two examples of what these settings might look like is below:

```
PATH=${Some Path}/jdk1.6.0_20/bin:${Some Path}/ant/apache-ant-1.8.1: ${More Path Info}
JAVA_HOME=${Some Path}jdk1.6.0_20
```

To verify that this is correct you will have to look at these values on your system One simple way to check the JDK version that you have is to run:

```
java -version
```

to see which one is in your path, and it should be a JDK 6 version to run this lab.

Please note that having an existing CLASSPATH environment variable set may cause odd issues with jar class loading, it is recommended to have this empty and not set. Please make sure to back up this value for when the lab is over. You are welcome to not do this, however weird things may happen when you are running through the labs.

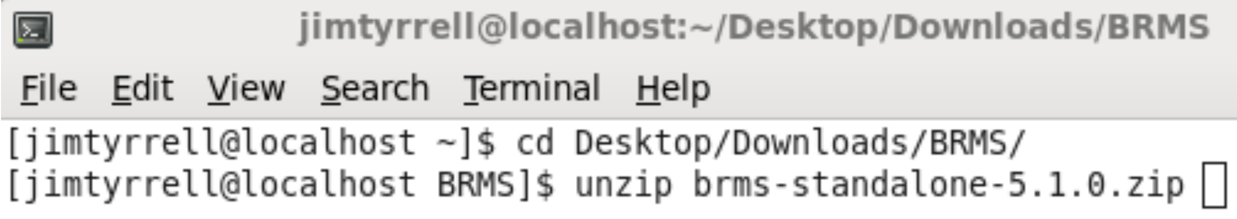
What is Expected of You

Please feel free to raise your hands with any questions that you have about the lab; feel free to ask why it is you are doing something, or if something does not feel right. Please know that all care was made in creating this user guide, but all screen shots and steps along the way might be off by just a little so please be patient with any issues.

Lab Number 1: Installation of BRMS

Get the File

In the ${USER_HOME}Downloads/BRMS directory you will find the BRMS installer, it should for Linux look something like this:



Run the commands after opening a terminal window:

```
cd Desktop/Downloads/BRMS
unzip brms-standalone-5.1.0.zip
```

Thats it you have the BRMS installed and almost ready for use.

Lab Number 2: Installation of JBDS

Get the File

In the ${USER_HOME}Downloads/JBDS directory you will find the JBDS installer, it should for Linux look something like this:

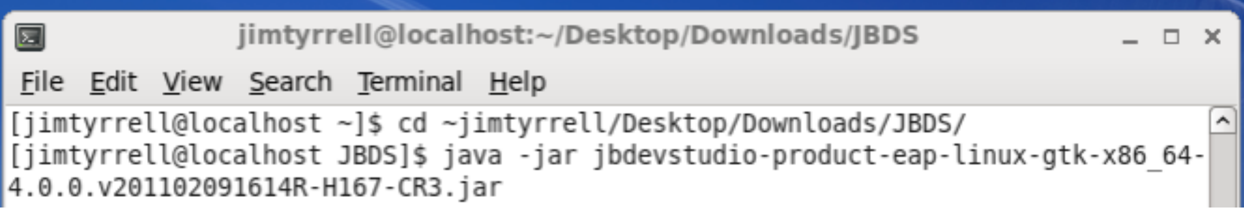
```
jbdevstudio-product-eap-linux-gtk-4.....
```

Running the Installer

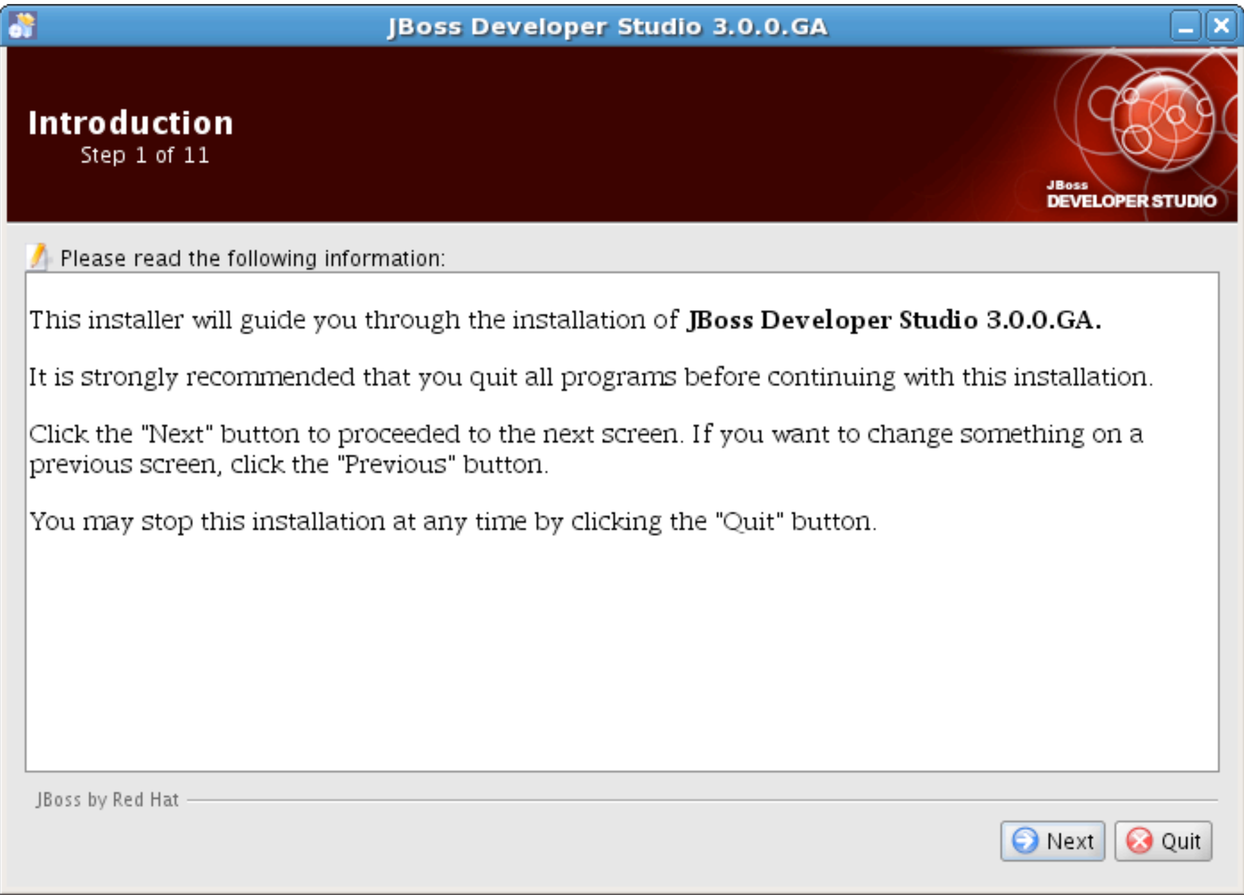
The next step is to run the installer. The command to do that is very simple:

```
cd ~student/JBDS
type in java -jar and j and the tab key to bring up the correct file
java -jar ${The_File_Available_For_Linux_Above}
```

A command prompt will be used for this and on Linux it looks like this



Running this command will then get you to the next step in the process the visual installer. It will look something like this:

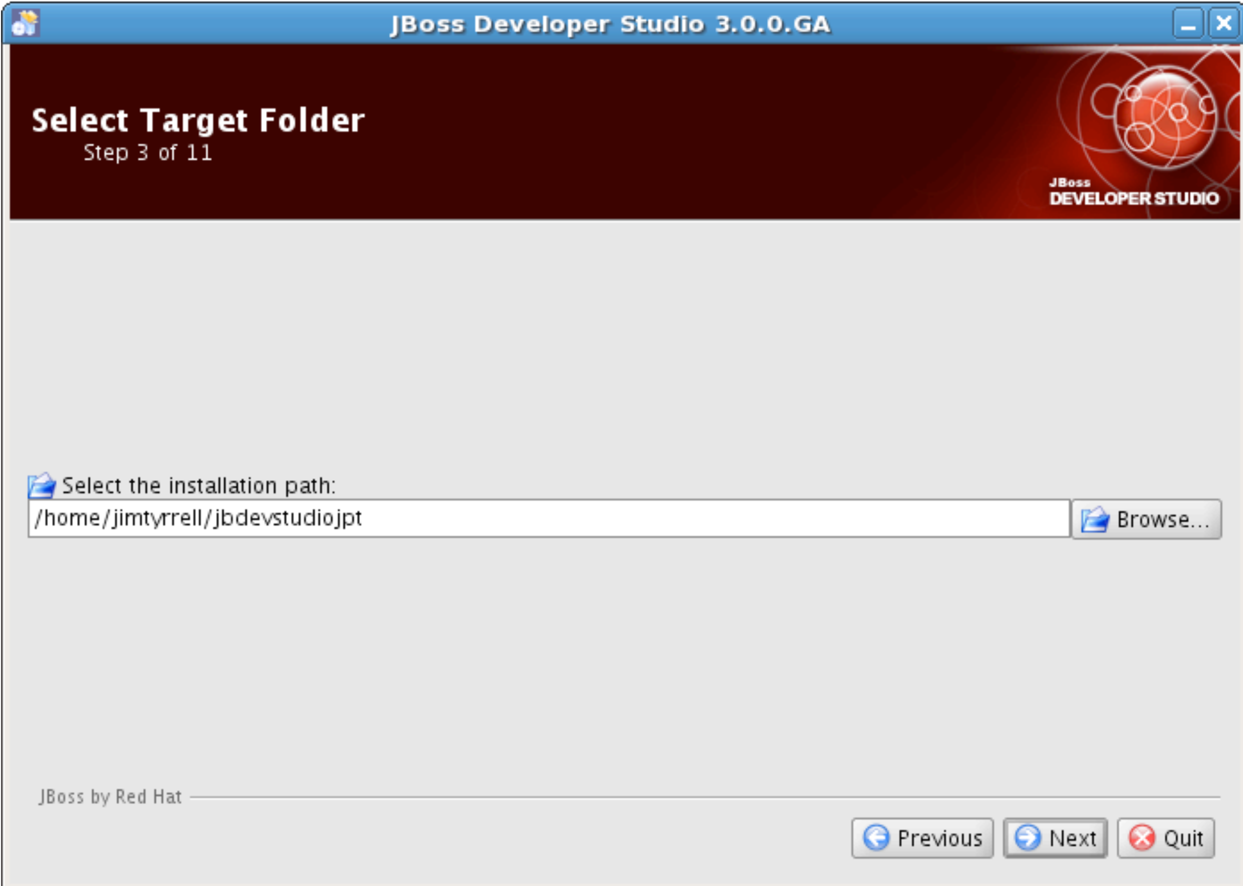


Please Select Next

Select the Next Button and this will get you to this screen:

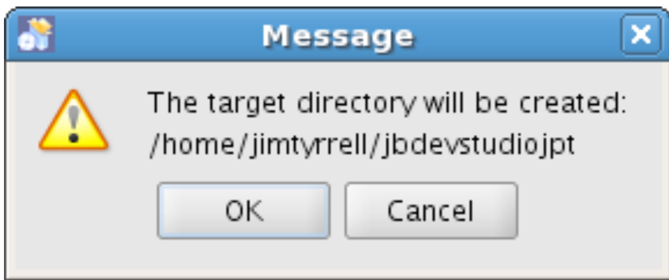
Please select the “I accept....” radio button to enable the next button. Click the next Button.

The next screen prompts you on where to install JBDS. Make the name unique and don't just select the default. A few recommendations. Make sure you write down or remember this path. Please make sure it does not have spaces in it, as older versions had problems running with spaces in the installed path and it should be fixed, but you never know. Also if you already have an existing copy of JBoss Developer Studio installed please raise your hand. Also please put in something unique in the name so that someone else after you will able to install this lab without any problems.



Please Select Next

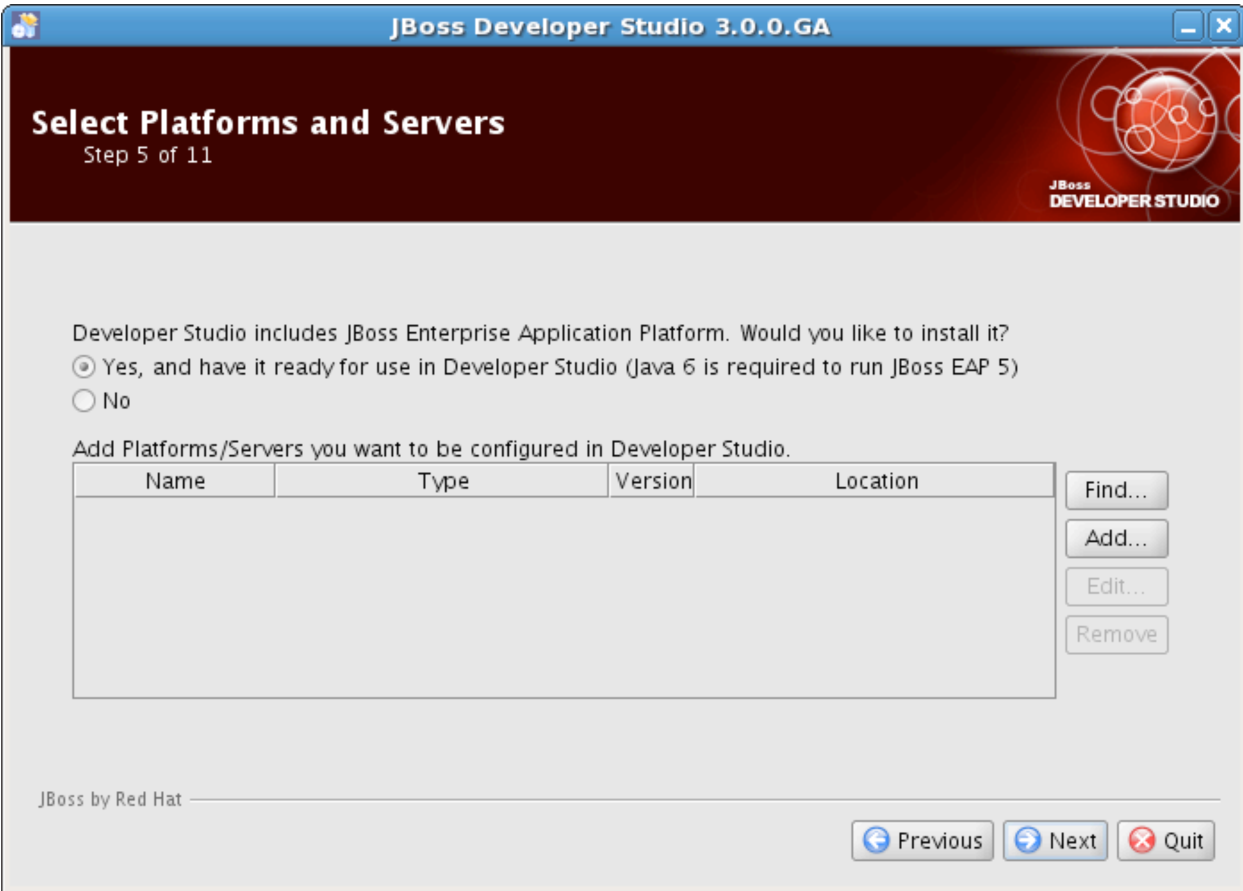
No changes probably will be made to this setting please select Next. You will be prompted to make this directory if it does not exist. If you do not see a prompt like this, select previous and make the installation path unique.



Next you will be prompted to use the JDK that was used from your path/java_home properties. No changes are needed with this setting.

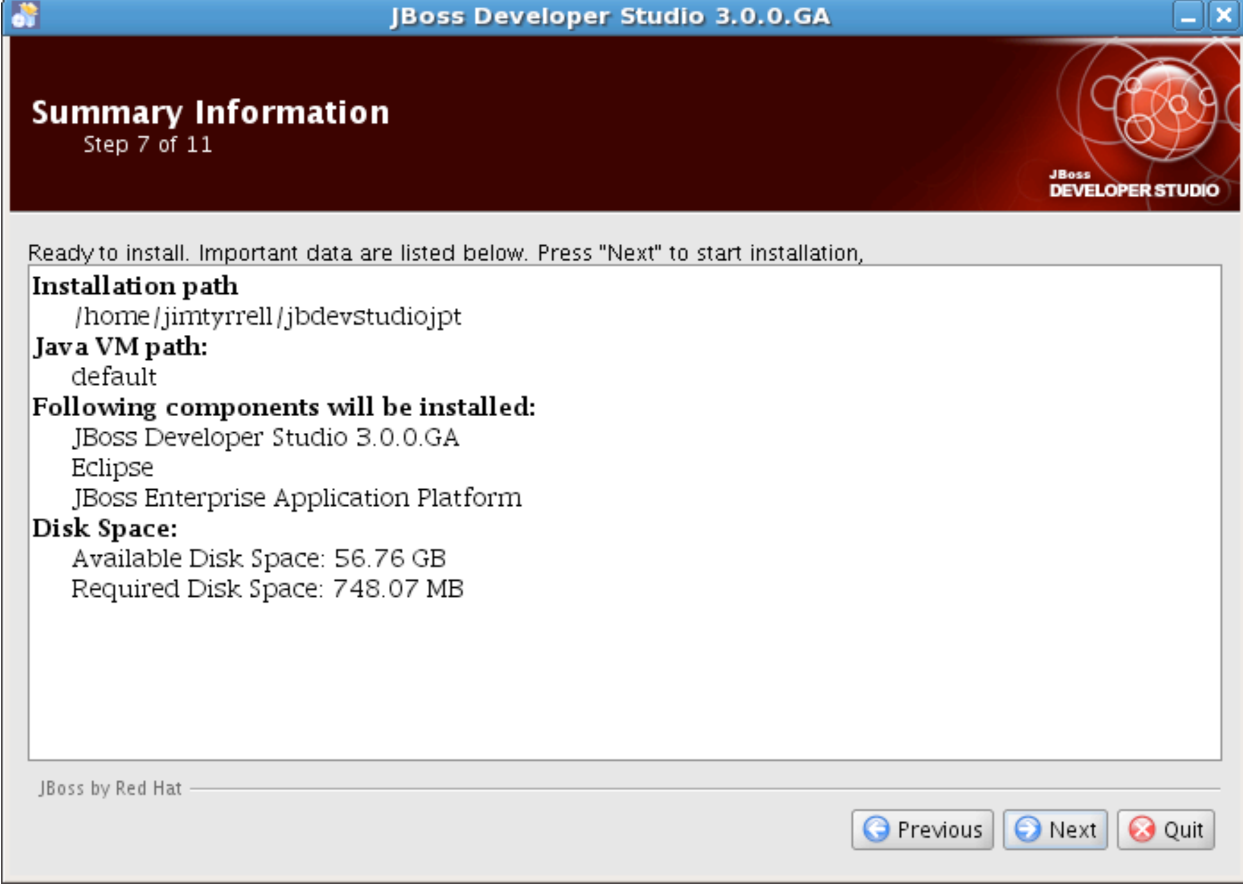
Please select Next

The next box will prompt you to install and make available the embedded EAP instance. Please leave this section alone and click Next. If this was the SOA-P (Service Oriented Architecture - Platform) Lab then you could add a SOA-P instance that would already have to be installed. You also could add another JBoss EAP/AS instance, but for the purposes of this lab this is not required and not recommended.



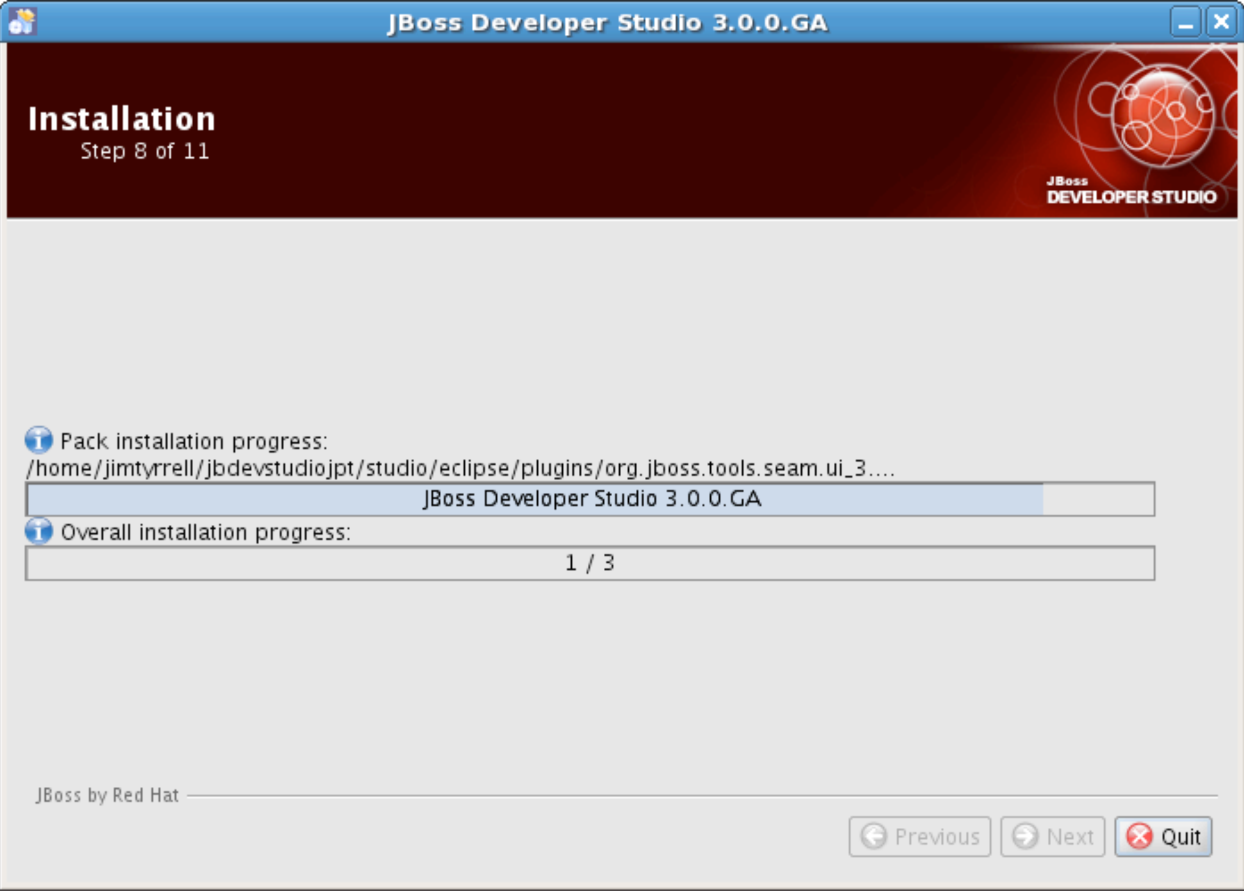
Select Next

The next screen just shows you what will be installed:



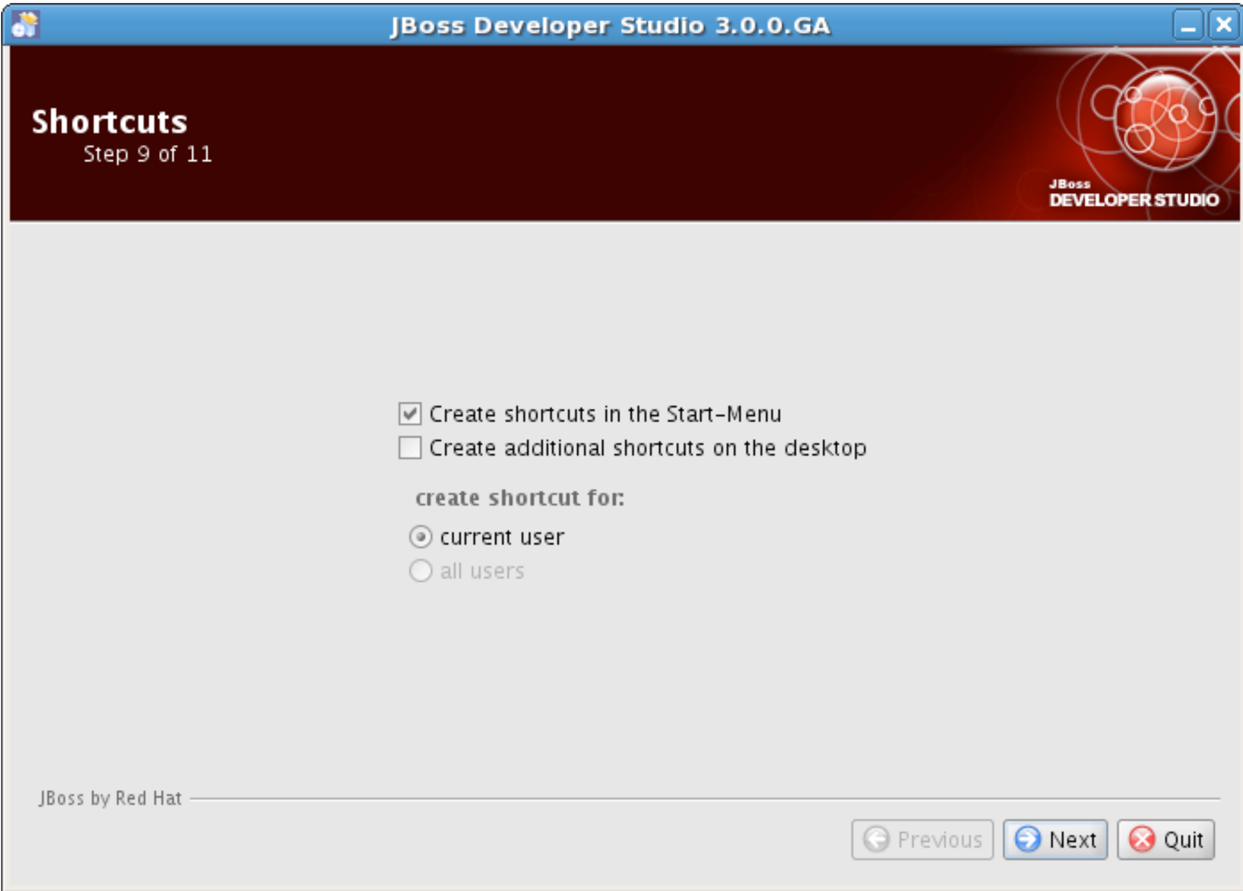
Click Next

The installer running



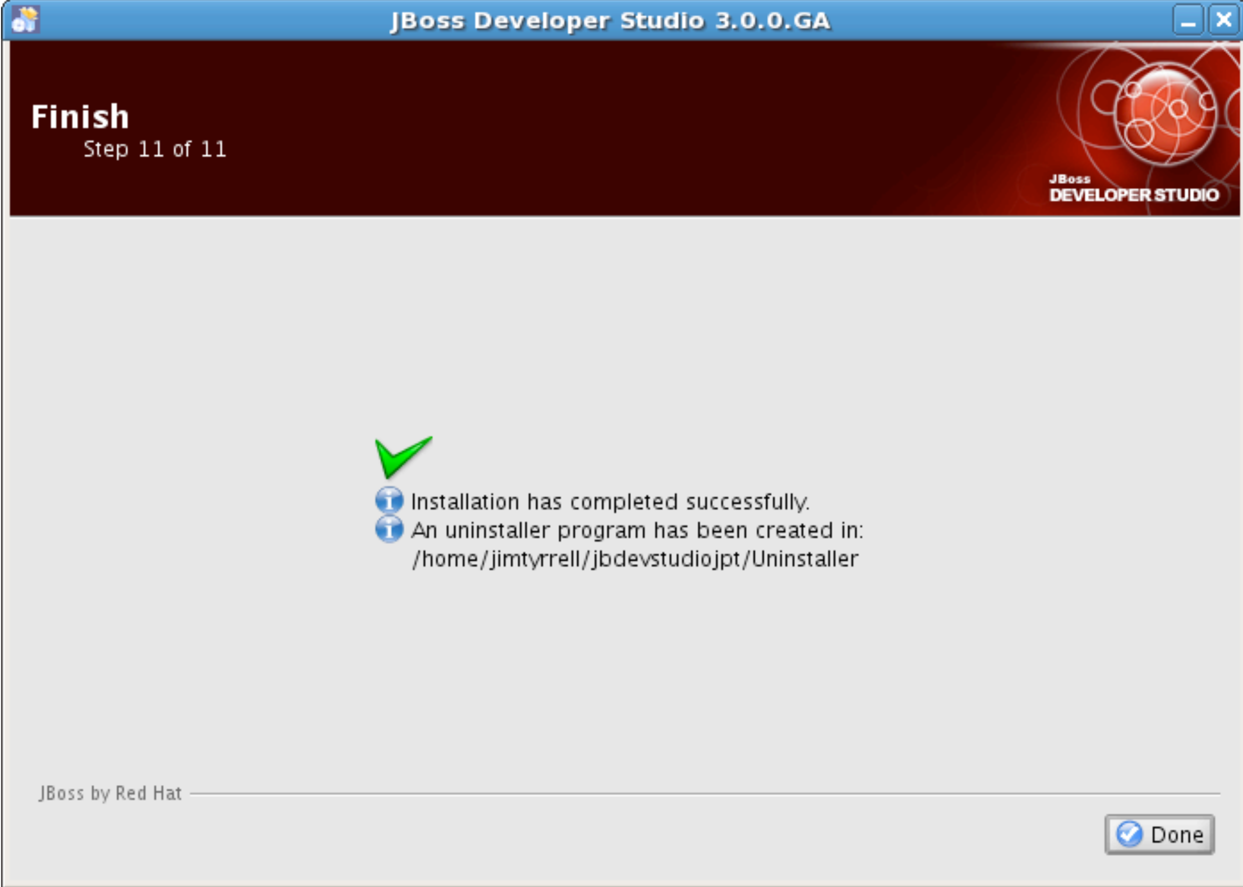
Wait for Next to be available to you once the installer is completed

The next step will show you option on where to place short cuts and the like. The defaults are fine for this lab.



Select Next if you want to make any changes, feel free.

Congratulations you have installed JBDS 2.0



Please Select Done.

Congratulations you are now done installing JBDS 3.0 on your local workstation. You have now completed this lab and we will next create a new Seam Project using the already running HSQLDB.

Lab Number 3: Start the BRMS

Config File Change

You will need to change to the directory below, and edit the following file via the nano command as shown.

```
/home/jimtyrrell/Desktop/Downloads/BRMS/brms-standalone-5.1.0/jboss-as-web/serve
r/default/conf/props
[jimtyrrell@localhost props]$ nano -w brms-users.properties
```

You will need to uncomment the following lines as shown:

```
jimtyrrell@localhost:~/Desktop/Downloads/BRMS/brm
File Edit View Search Terminal Help
  GNU nano 2.0.9            File: brms-users.properties

admin=admin
#mailman=password
```

Make sure you save the file

Start BRMS

In the ${USER_HOME}Downloads/BRMS where you unzipped the brms*.zip file, you will need to change to the:

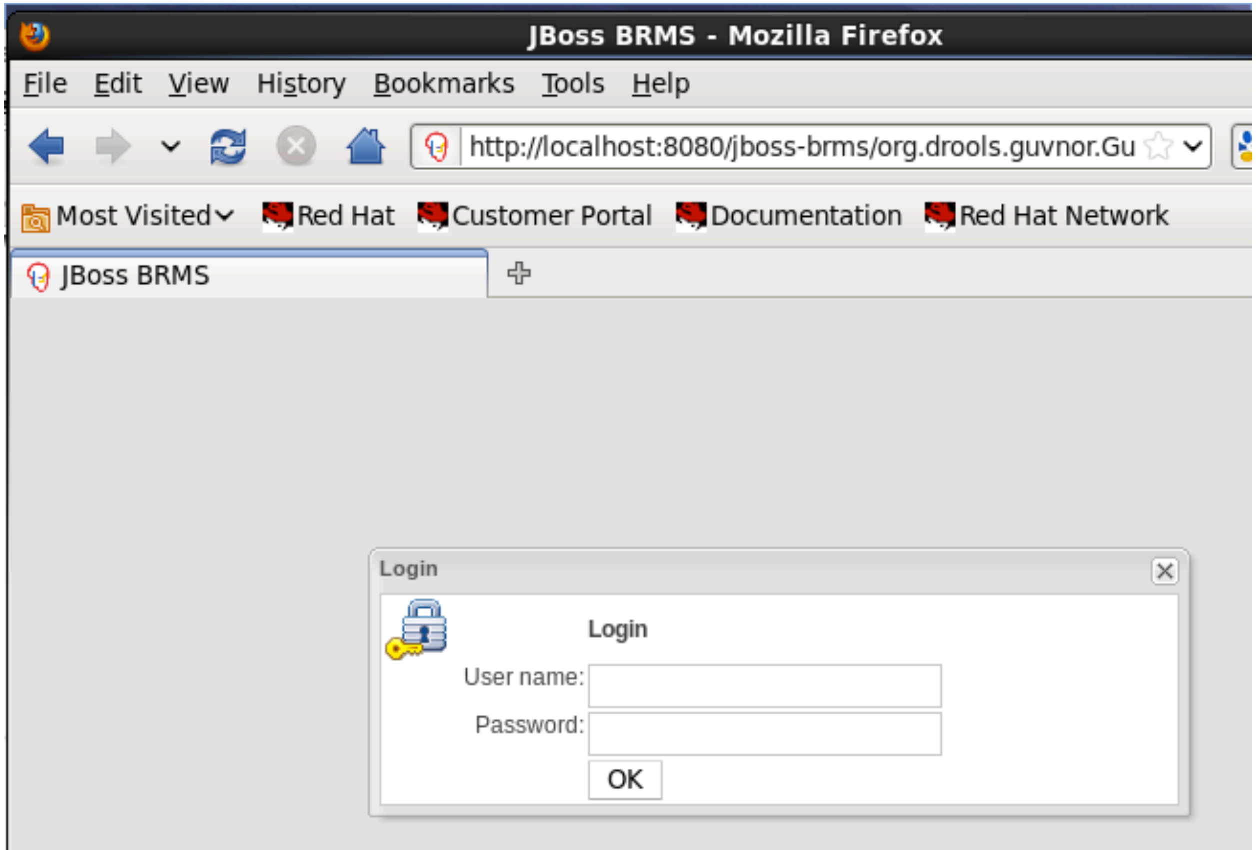
```
cd brms-standalone-5.1.0/jboss-as-web/bin
./run.sh -c default
```

```
[jimtyrrell@localhost BRMS]$ cd brms-standalone-5.1.0/jboss-as-web/bin/
[jimtyrrell@localhost bin]$ ./run.sh -c default
```

Wait for it to finish starting, you will see a message like this:

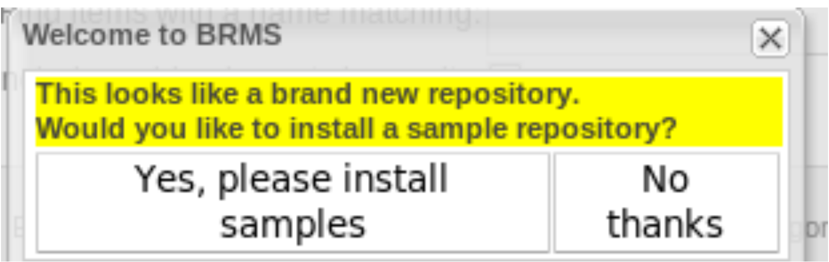
```
19:57:46,760 INFO  [Http11Protocol] Starting Coyote HTTP/1.1 on http-127.0.0.1-8
080
19:57:46,773 INFO  [AjpProtocol] Starting Coyote AJP/1.3 on ajp-127.0.0.1-8009
19:57:46,777 INFO  [ServerImpl] JBoss (Microcontainer) [5.1.0 (build: SVNTag=JBP
APP_5_1_0 date=201009150134)] Started in 34s:589ms
```

That's it, now open up a web browser, and look at http://localhost:8080/jboss-brms:

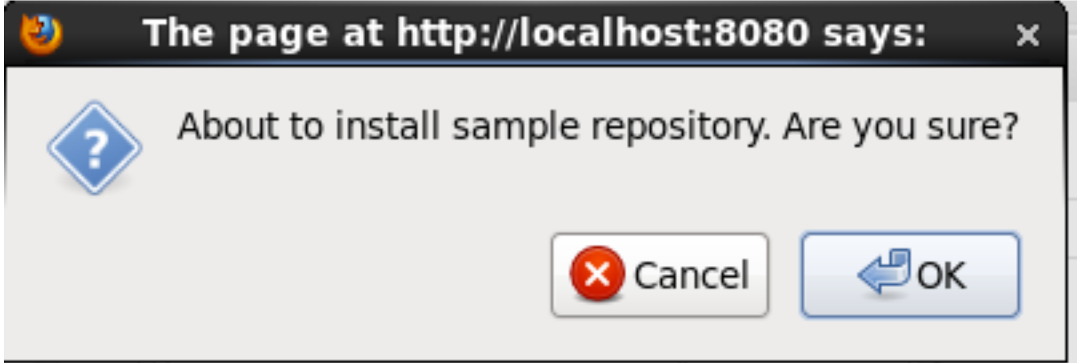


Now login with admin/admin

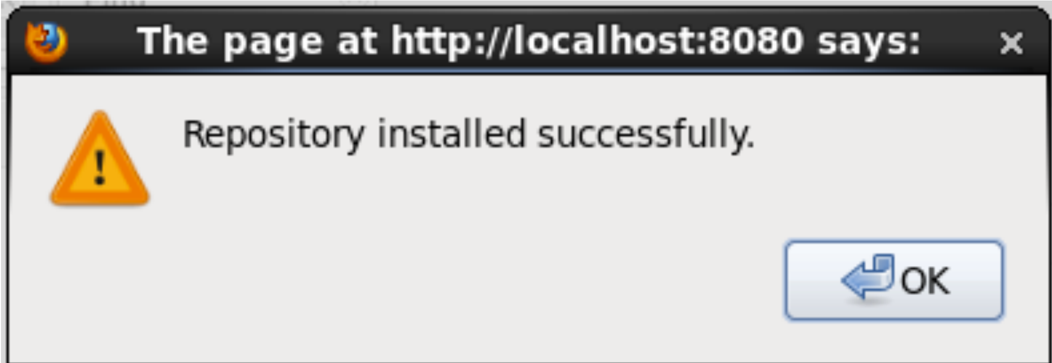
Now you will be prompted with the following:



Click on "Yes, please install samples"



Click on “OK”

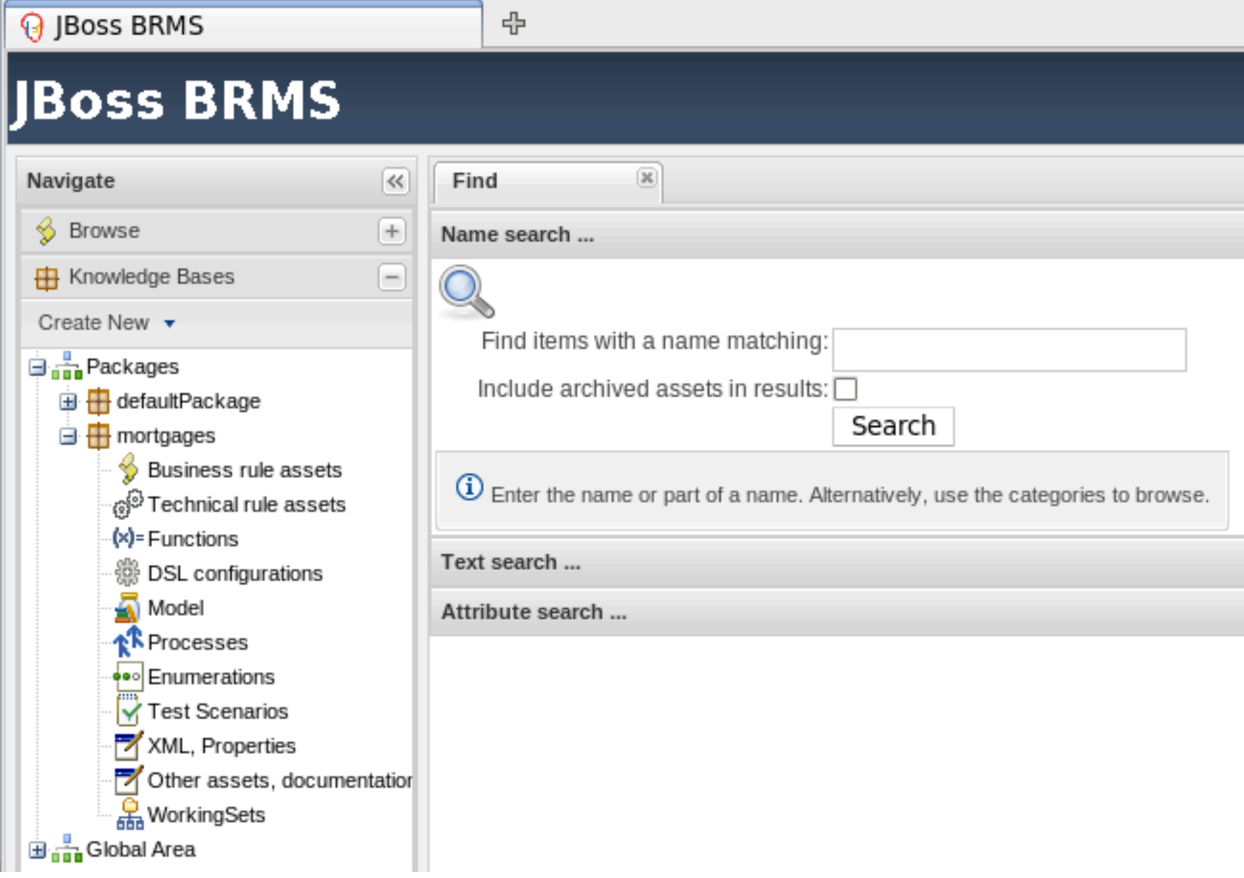


Click on “OK”

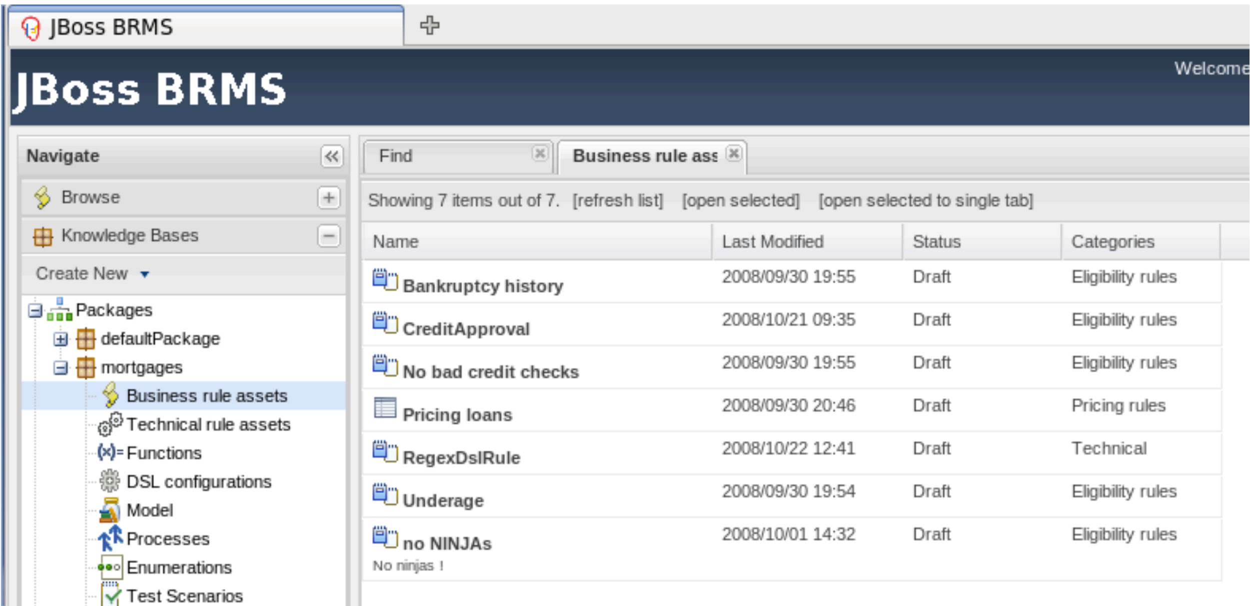
Lab Number 4: Explore the Sample Application

Explore the Sample Application

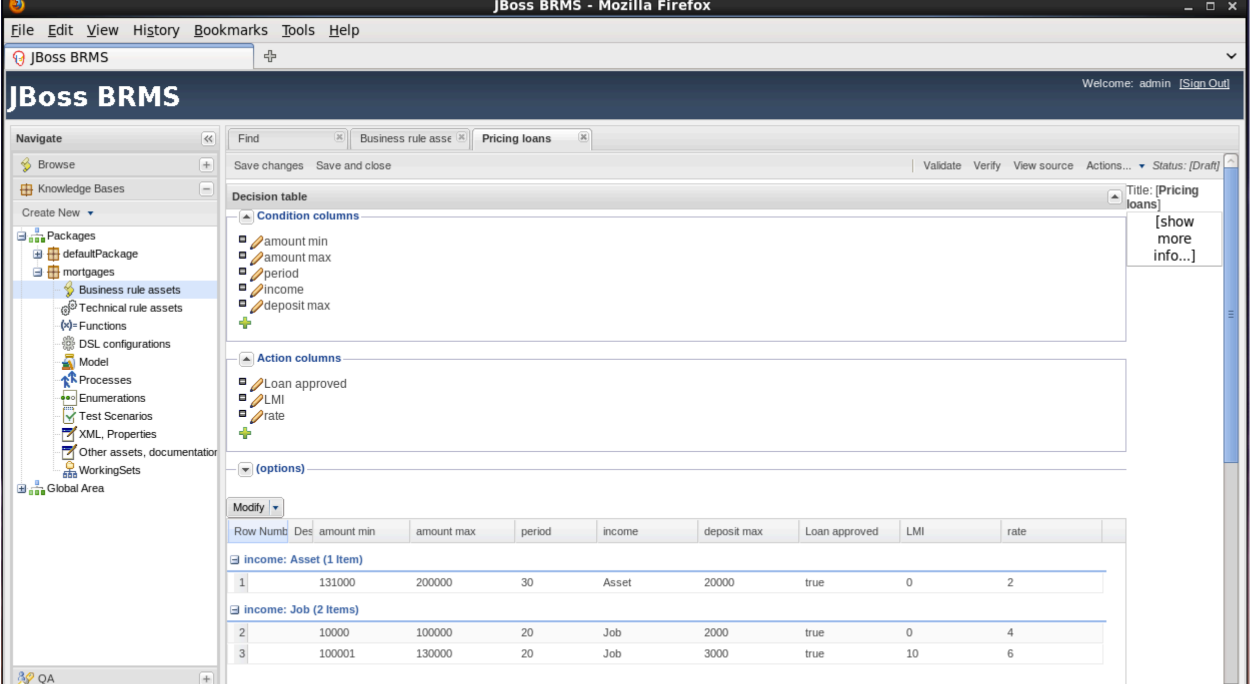
Open up the Knowledge Bases and expand out the Packages -> mortgages as shown:



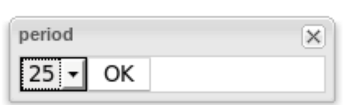
Click on the Business Rule Assets



Double Click on Pricing Loans, this will open a decision table, essentially a spreadsheet for storing tabular data that will be used by the rule engine. In this case it maps incomes, loan requested, down payments, etc to the willingness of the organization to make a loan. Click the downward facing arrow in the upper right corner (You might need to expand your browser window) shows you the mappings between the Java Objects and these tables. Click on the pencil icon to the left of amount min and you will see the following screen, and pop up box/window. Notice some of the mappings, we will expand these java objects in a bit:

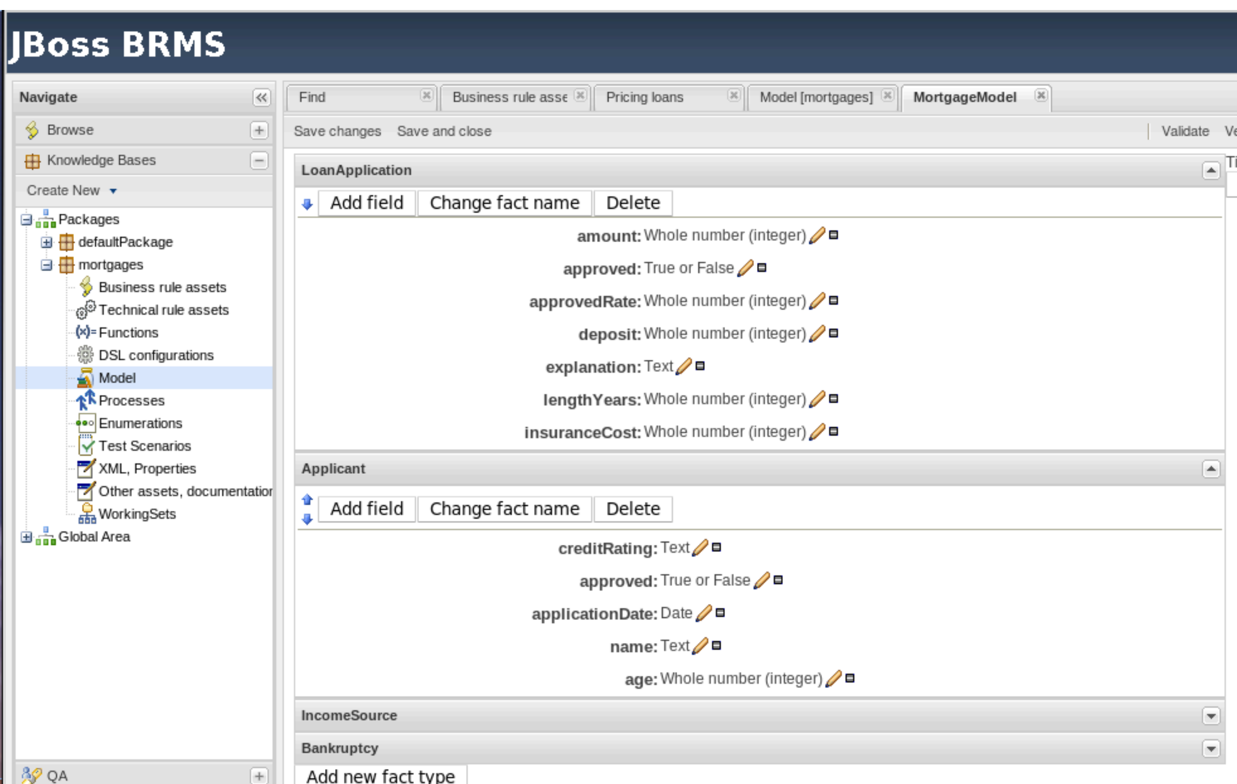


Click the x to close the pop up box. Click the now upward facing arrow to minimize those mappings, and now notice the highlight of row 2 under income and a click on the Modify arrow as shown, this will be important a little later on in the lab:



Click on the 20 in Row 2, and update the 20 to a 25, notice that an Enumeration was provided for you. Change this back to 20. Then please click on "OK".

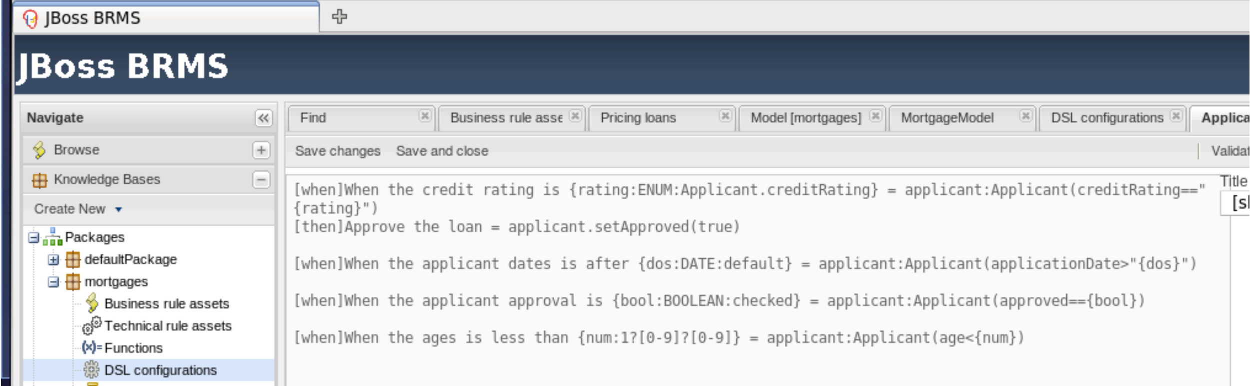
Lets explore some of those mappings in a little more detail. On the left hand side click on Model, once that opens click on MortgageModel, then click on the downward facing arrows to expand "LoanApplication" and "Applicant" and this will bring you to the screen as shown, note that these are essentially Java Beans that are just objects that store data:



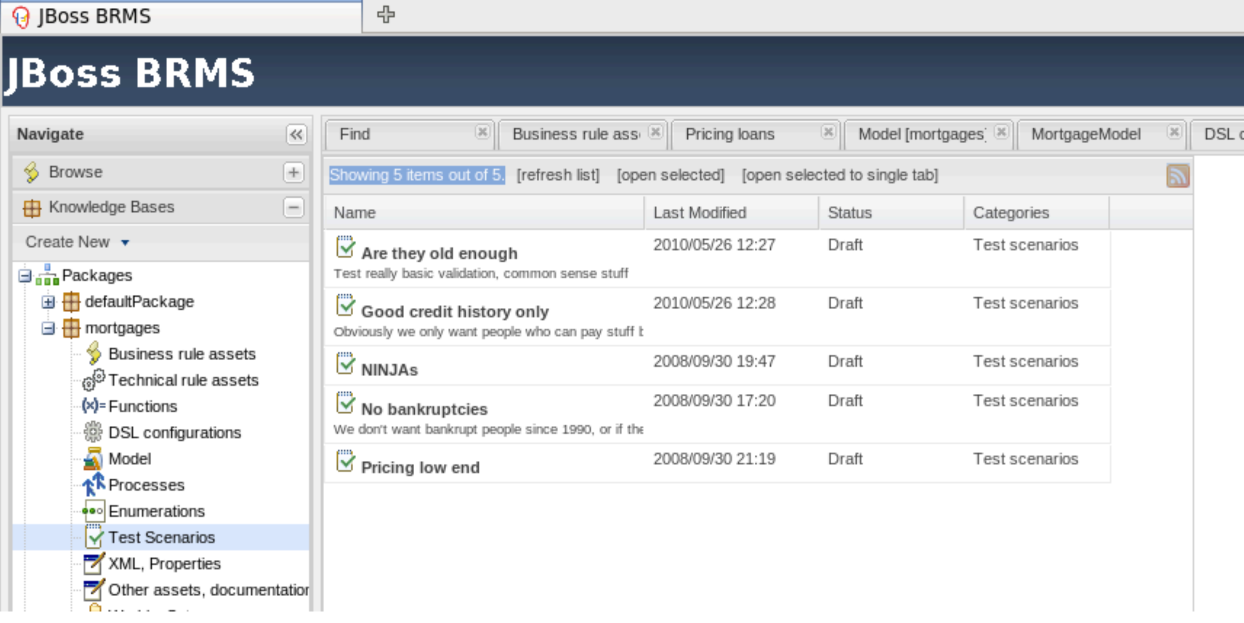
Expanding the Loan Application by clicking the downward facing arrow, and viewing the Member Fields of the LoanApplication Object that is being created for you under the covers. In other words this is our Domain or Object Model, and in Rules area, we call these Facts, simply Java objects that we previously viewed the mappings for with the Pricing loans Decision table earlier.

Explore the Domain Model Objects and the mappings in Pricing loans by clicking the tabs above. Once you think you have a good handle on the model and the mappings, we can move on to another part of the user interface. If you have any questions please raise your hand.

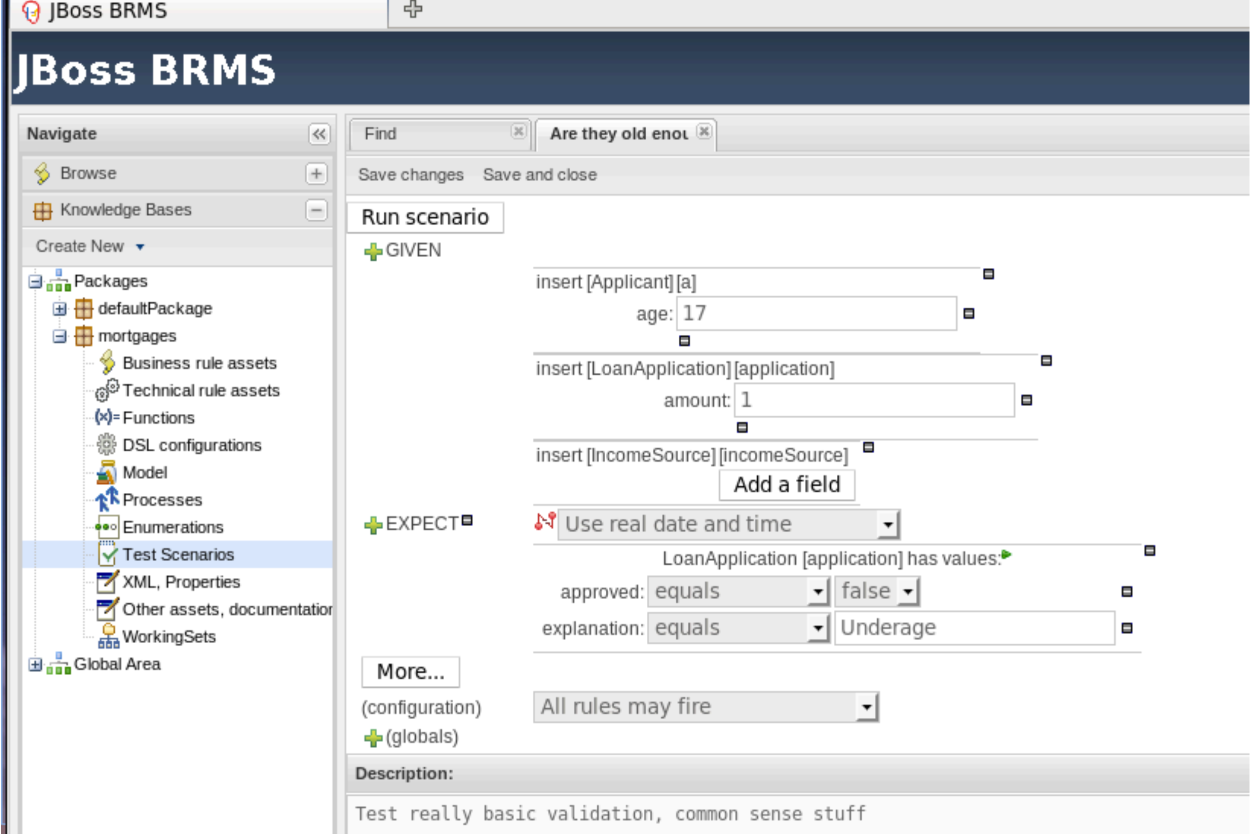
Go back to the Business Rule Assets, and look at Underage. This is an example of the DSL (Domain Specific Language) that is used to guide a business user or analyst into making some decisions or putting in correct values when writing a rule``. You can see the DSL file, via DSL configurations, and clicking on ApplicantDSL, you should see this:



Now that we have explored some of the assets that make up rules, we need to explore what it takes to test the rules. Click on Test Scenarios and you should see something that looks like this:



Clicking on Are they old enough brings up a screen like this:



Clicking on Run scenario, runs the test case against the Rules engine. You should see a green bar indicating success. This is not unlike jUnit or TestNG if you are familiar with them. A simple test case to validate that with a given input you have the proper output when running that input through the rules engine. Lets try a few things:

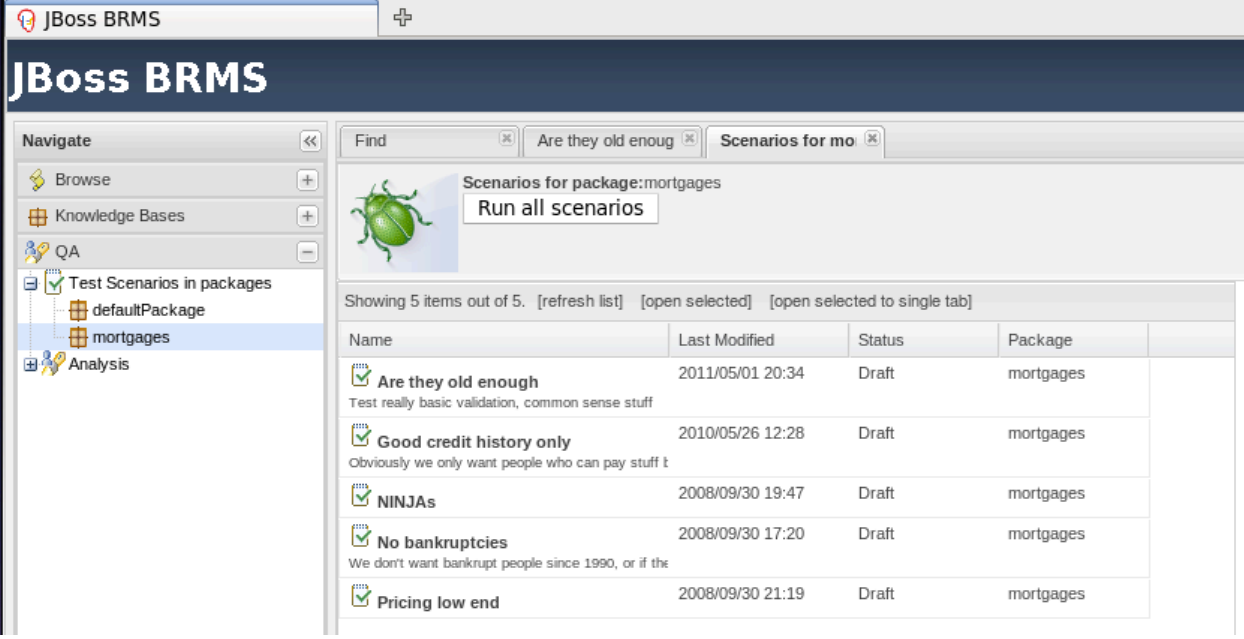
1. Change the age to 22, and run the test, what happens?
   Did you notice that the green bar turned to 0% coverage, two expectations were not meant.
2. Click on the Show Rules Fired, to view the Rules that were triggered. See which ones were triggered
3. Now change the age back to 17, and run the test again. Did you notice what happened?
4. Now Click on Show Rules Fired and see how the Rules fired

If you wanted to change the age to 18, lets do that, and to make sure it takes effect, make sure you click Save changes to make this change available

You now have explored some of the assets that make up the BRMS, and you have now completed this lab.

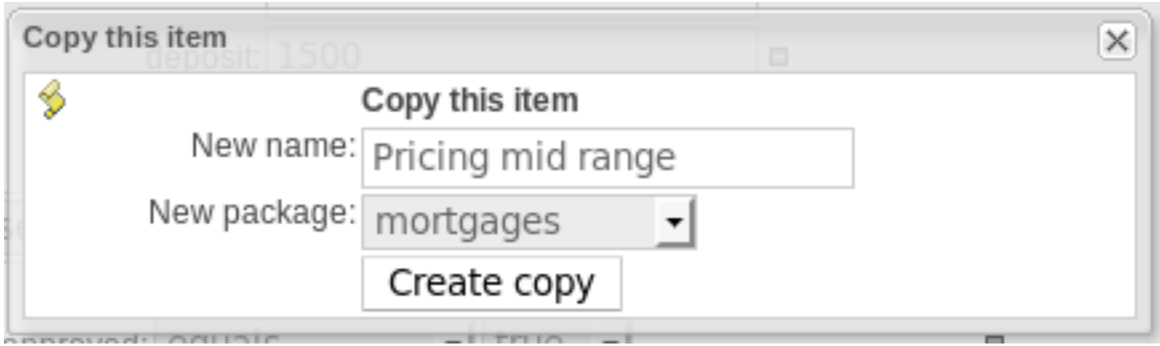
Lab Number 5: QA and Testing

Open the QA tab in the same way that you expanded the Knowledge Bases earlier, Expand Test Scenarios in Packages, and click on mortgages as shown below:



Now click on Run all scenarios at the top of the page. You can see that we have no missed expectations, such a nice feeling, something one never gets often enough. Note however that we have two uncovered rules. Looks like to me we need to create a test case for Row 2 and Row 3 for Pricing loans. Lets do that, create two new test cases.

Lets open the Test Scenarios again under the Knowledge Basis, and open up the Pricing low end, and make a copy of it. You do this by Clicking “Actions” -> Copy in the upper right hand corner. Lets call it pricing mid range:



In a perfect world a QA person or business analyst would have to be the one creating these test cases, as it is not a best practice to have the rule creator, be the rule tester.

Pricing mid range will be open for you, and update the Loan Application->amount to 120000, insuranceCost to 10 and approved Rate to 6 as shown:

Run scenario

Results: 100%

Summary: [application] field [approved] was [true].
[application] field [insuranceCost] was [10].
[application] field [approvedRate] was [6].

Audit log: Show events

GIVEN

insert [IncomeSource] [income]
type: Job
amount: 65000

insert [LoanApplication] [application]
amount: 120000
deposit: 1500
lengthYears: 20

EXPECT

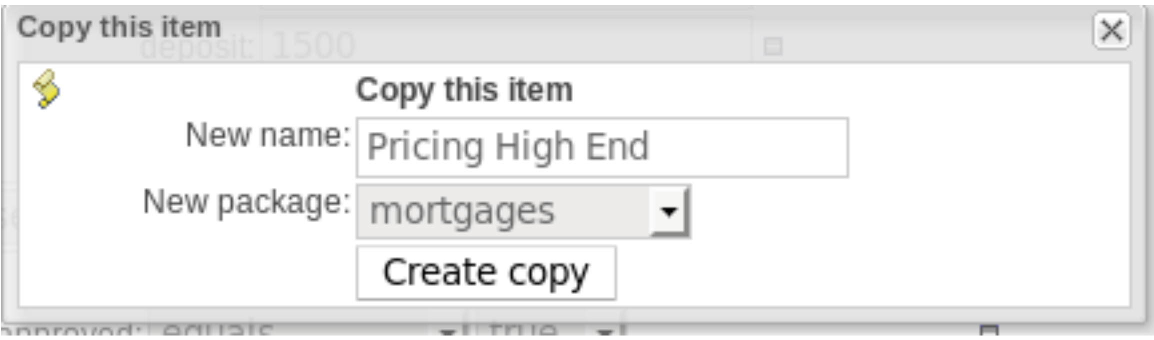
2 rules fired in 1ms. Show rules fired

Use real date and time

LoanApplication [application] has values:
approved: equals true
insuranceCost: equals 10
approvedRate: equals 6

Click Run scenario and you should notice that that the test turned green

If you click on Show rules fired, you will notice that Row2 is now covered.



Lets do the same thing for covering/testing Row 3, copy Pricing the low end, call it “Pricing high range”. Make type “Asset”. LoanApplication amount be 140000, insuranceCost 0, and change the approvedRate to 2. Clicking Run scenario, you should see that you get a green bar.

Make sure you click save changes on on the two new test cases you created.

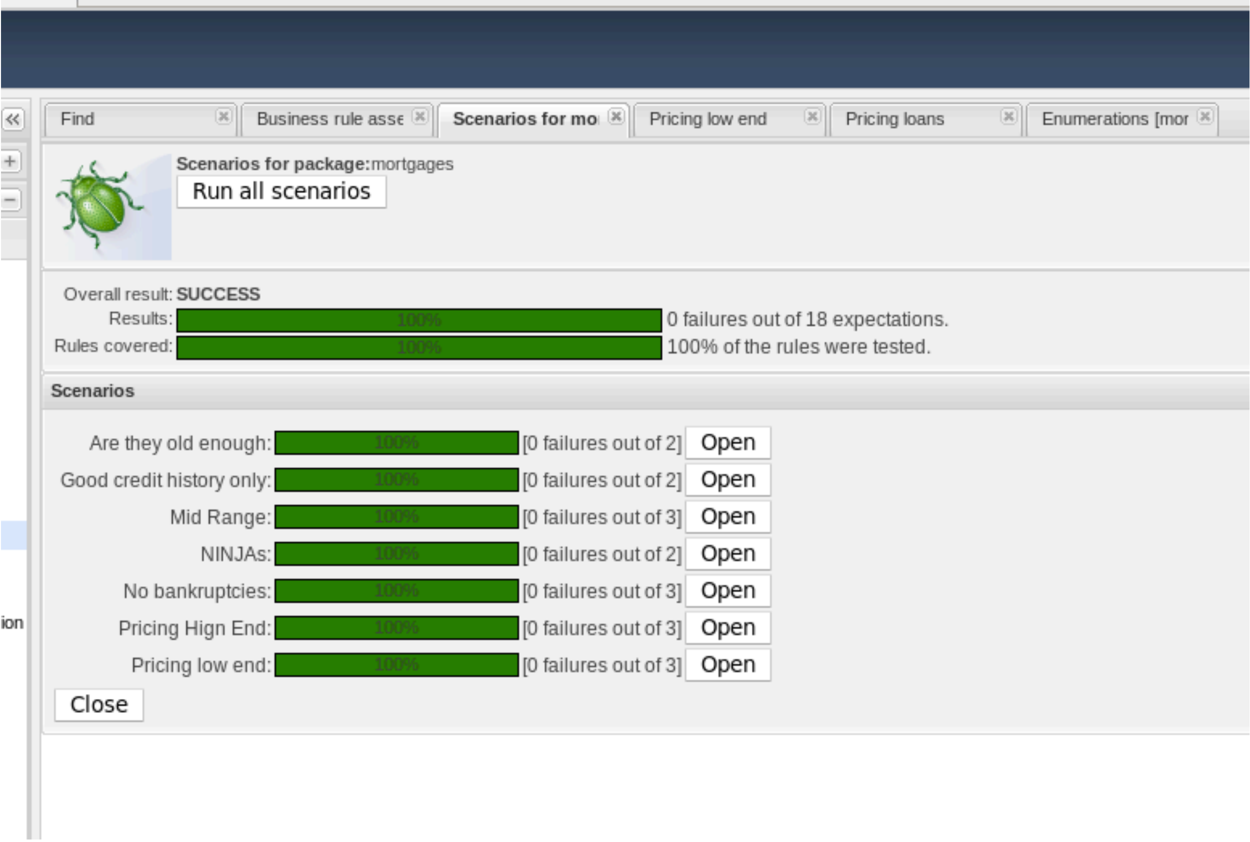
Make sure you save both the Pricing High End, and Pricing mid Range.

Go back to the QA -> Mortgages list, and click on Refresh List.

Now the question is, is this enough testing. If you click on Analysis on the left hand side of the tab, and click on mortgages, you will see a “Run analysis button. Click on this, an you will see some of the “gaps” that are in the test cases. The info presented here is beyond the scope of this lab, but it is something to be aware of in the power that is the BRMS. You will not it might present a lot of items, that may or may not be issues that need to be dealt with. A future enhancement is to mark items as ignored, but that is not available in the current product.

Click on Run all scenarios, as shown:

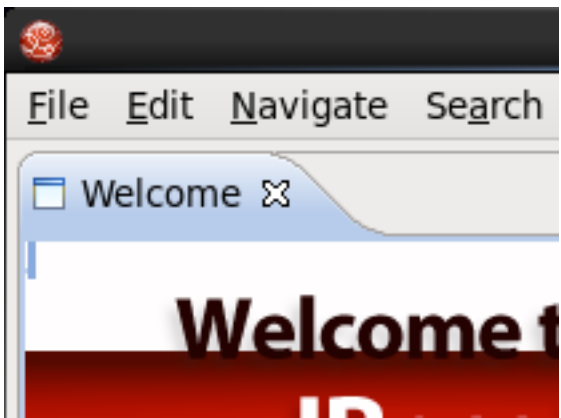
If we go back to the QA tab -> Mortgages and click Run all scenarios, you will see complete coverage, as shown:



Lab Number 6: JBDS Examples

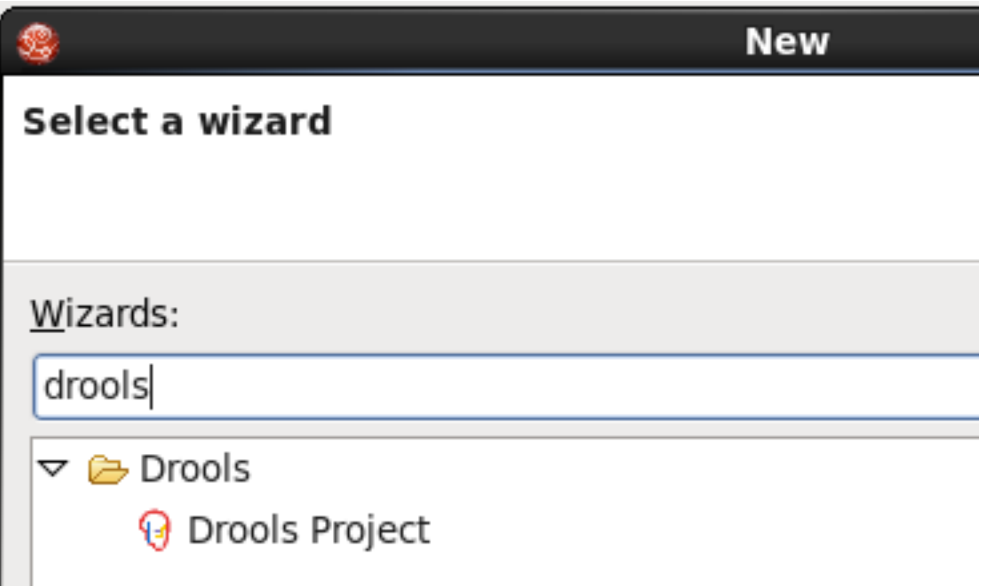
Open Application -> Programming -> JBoss Developer Studio 4.0.0
When it starts just say yes to the default workspace.
Feel free to to say yes to letting Red Hat look at how you use the JBoss Developer Studio.

Click the X to the right of the welcome screen as shown:



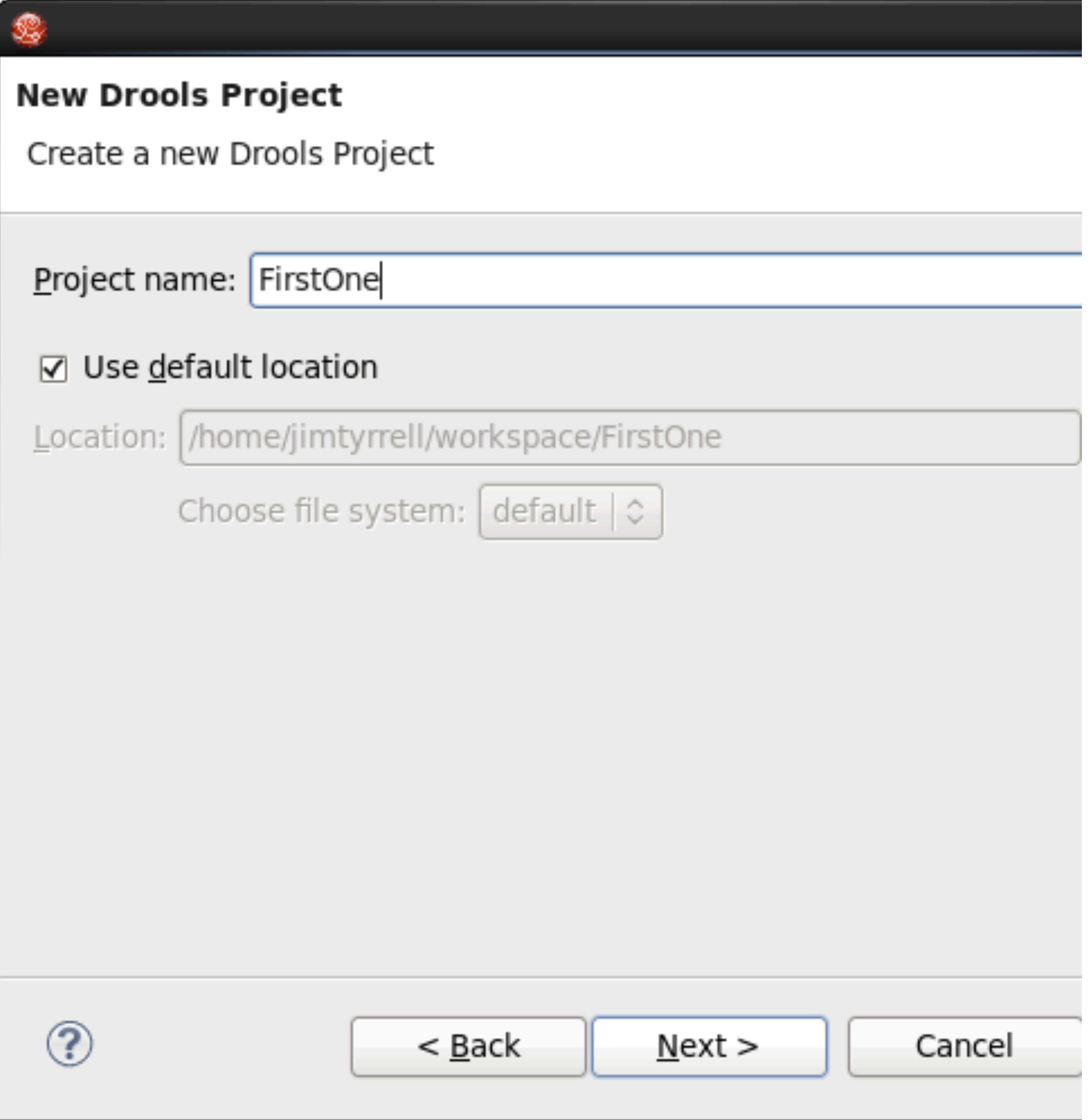
Open File -> New -> Other

Search on Drools as shown:



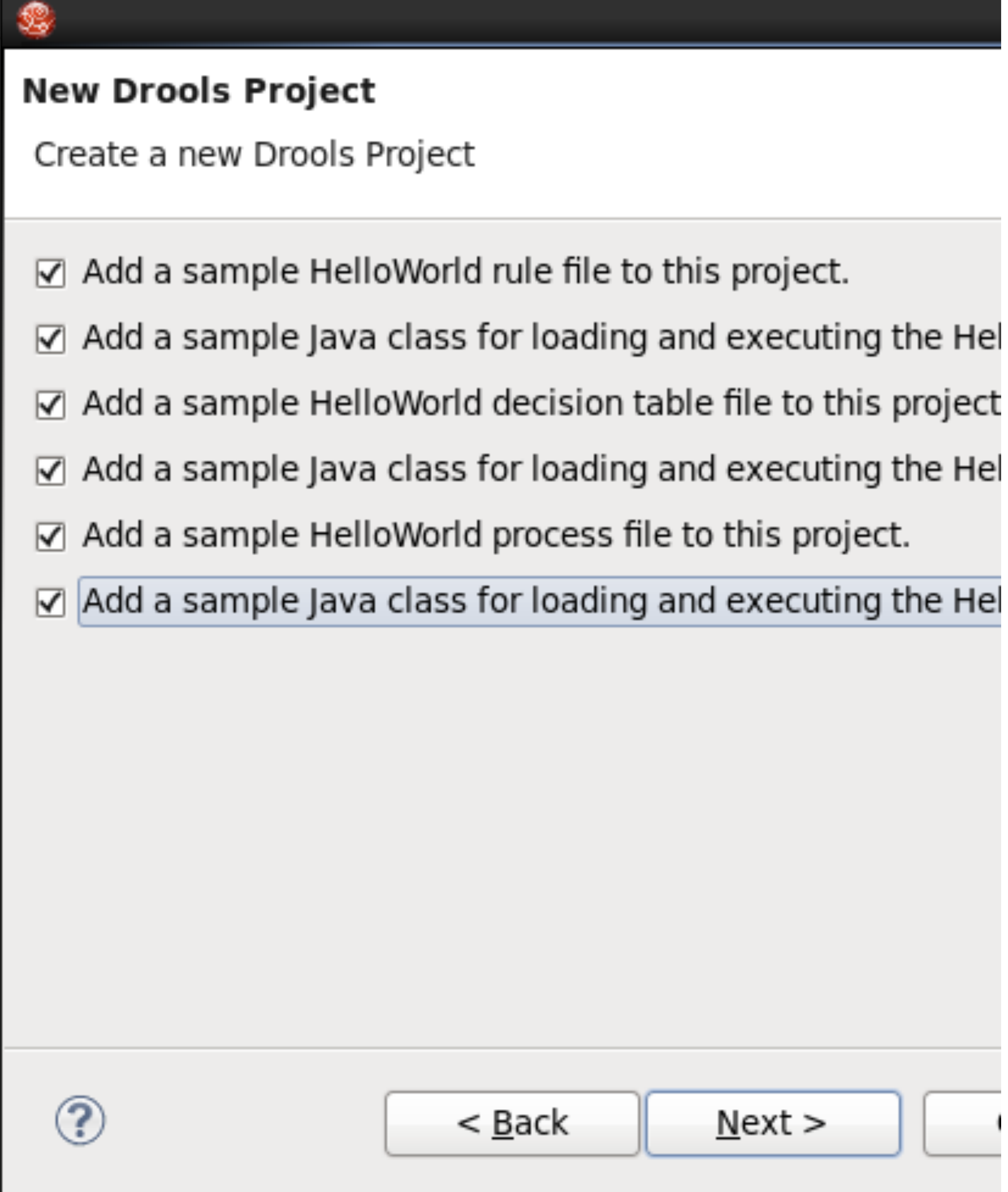
Click Next

Put in the Project Name, FirstOne as shown:



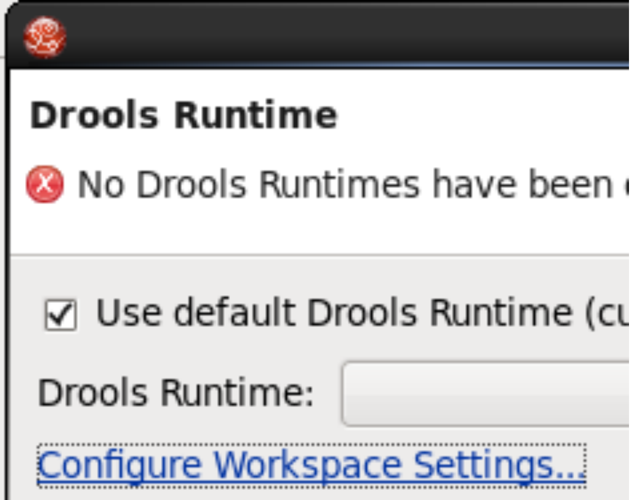
Click Next

Select all of the examples as shown:



Click Next

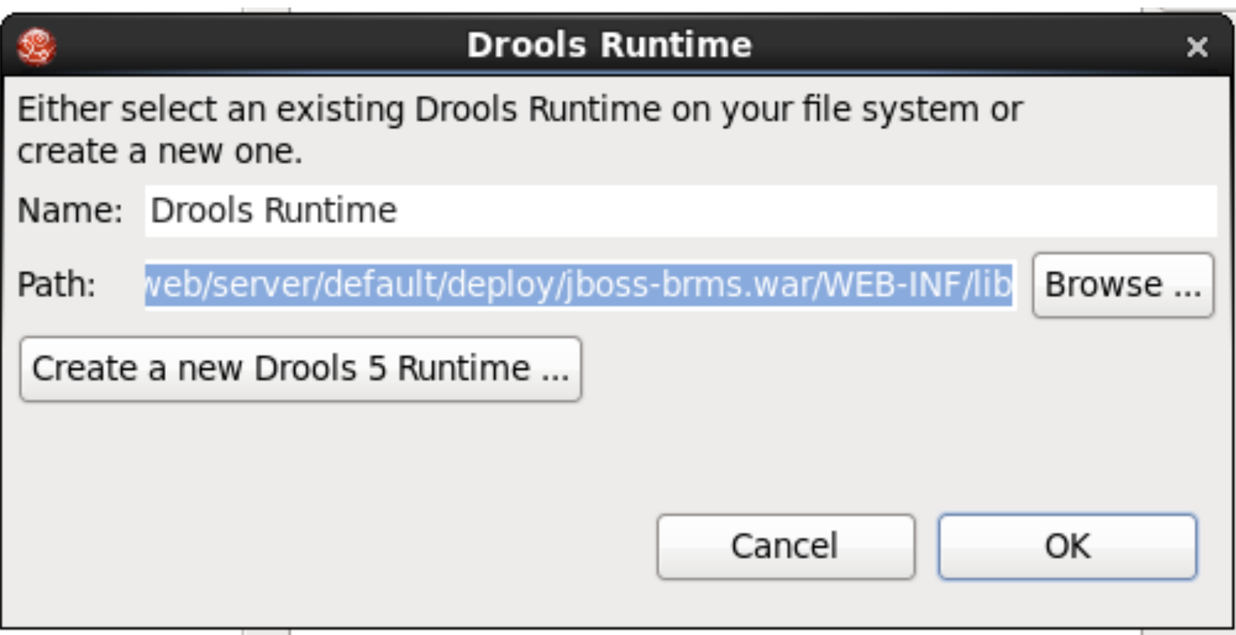
Now you will have to select the BRMS Drools runtime, click the "Configure Workspace Settings"



Click "Add" and navigate to:

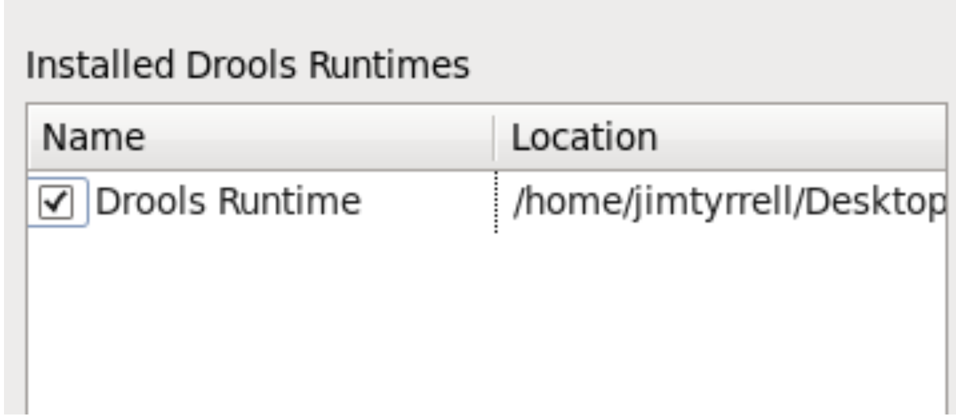
/home/student/Desktop/Downloads/BRMS/brms-standalone-5.1.0/jboss-as-web/server/default/deploy/jboss-brms.war/WEB-INF/lib

So it looks like this:



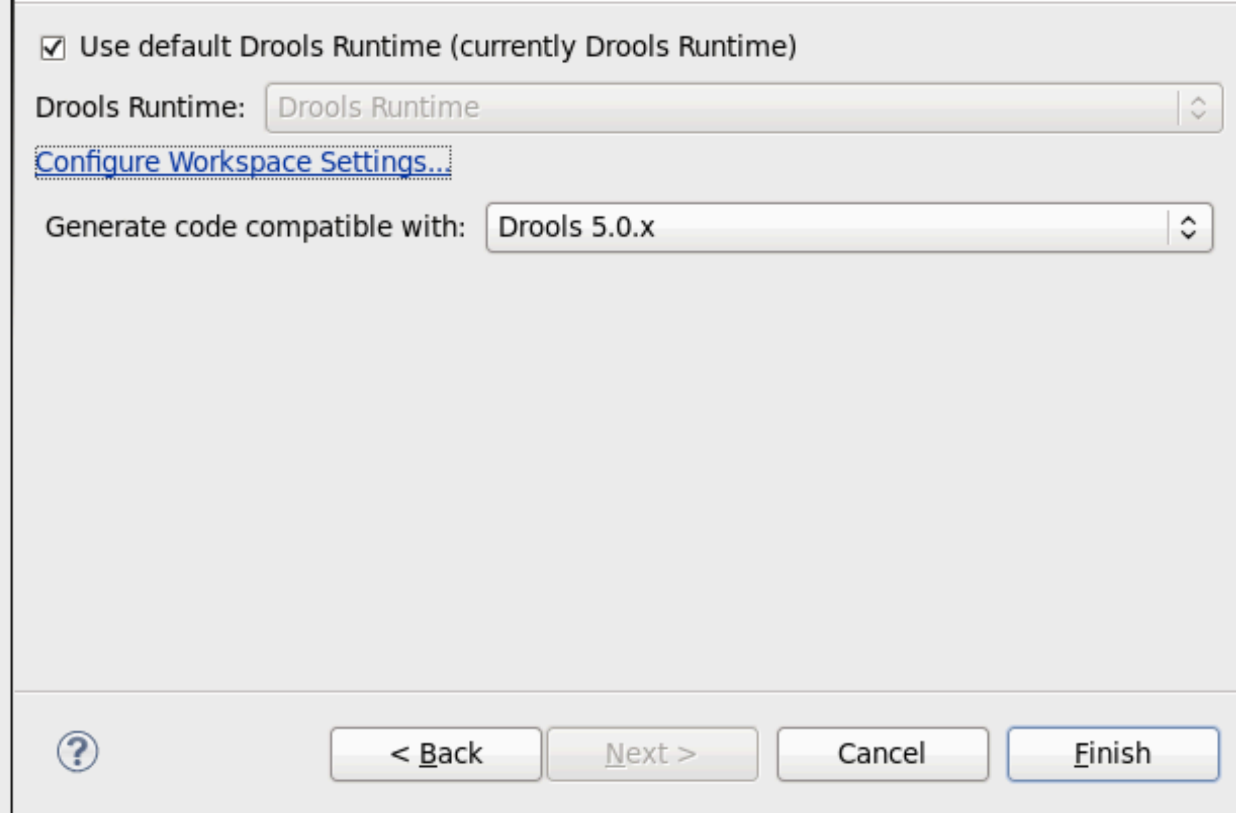
Click "OK"

Check the runtime as shown:



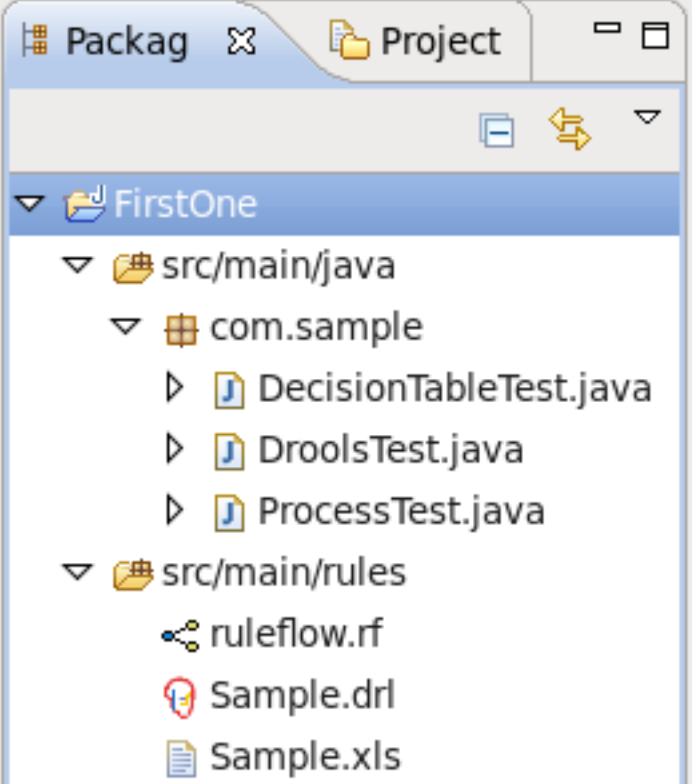
Click "OK"

You will now see that the Drools 5.x is available now:



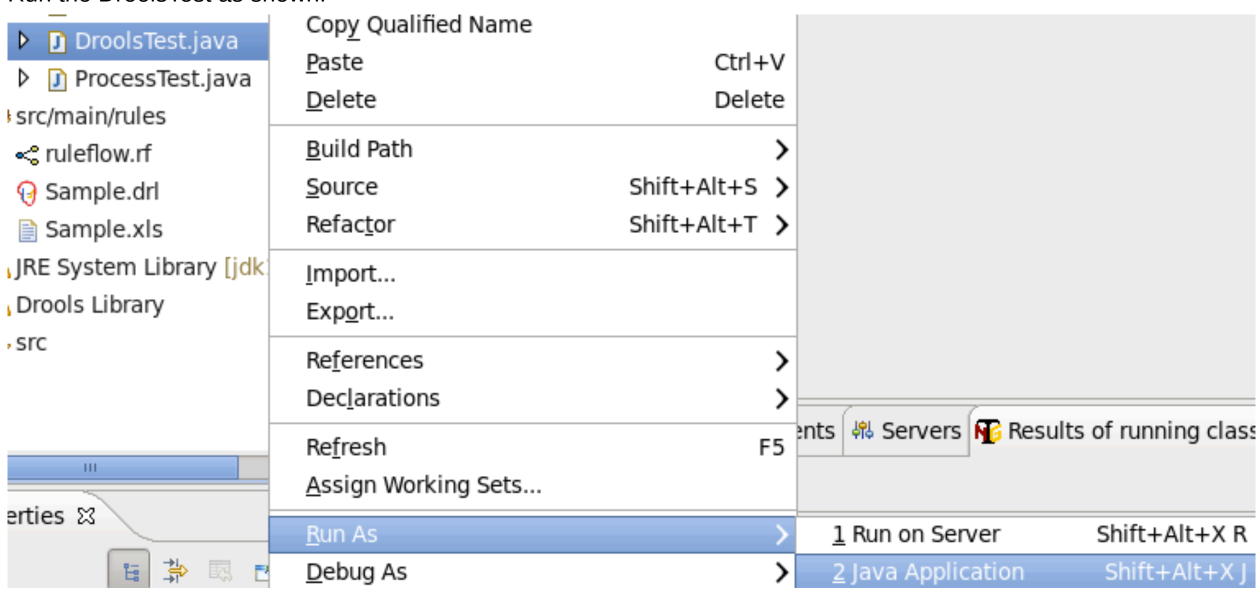
Click Finish

Expand out the src/main/java and src/main/rules

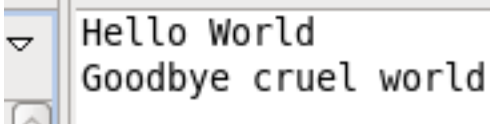


As shown.

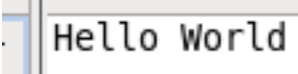
Run the DroolsTest as shown:



You should see in the console the following output:



Do the same with ProcessTest, you should see the following output:



So lets explore what is happening a little bit in that first example:
Lets open the Sample.drl file as shown:

Sample.drl

```
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( "This is it, Goodbye cruel world" );
		m.setStatus( Message.GOODBYE );
		update( m );
end

rule "GoodBye"
	when
		Message( status == Message.GOODBYE, myMessage : message )
	then
		System.out.println("My Name:  " + myMessage );
end
```

Text Editor | Rete Tree

“This is really it, ...” in the Goodbye cruel world line as shown in the first Rules “Hello World”, and “MyName: “ + myMessage in the the second one as shown. Save this off by clicking on the disk icon in the upper left corner and right click on the DroolsTest.java and run it again. You will see the above output in the console. Raise your hand if you are having any questions about what to change or what you should see.

Now lets see the source code that runs this, open up the DroolsTest.java as shown below:

```
Sample.drl    DroolsTest.java

package com.sample;

import org.drools.KnowledgeBase;

/**
 * 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.newStatefulKn
            KnowledgeRuntimeLogger logger = KnowledgeRuntimeLoggerF
            // 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();
        }
```

Lets explore this code a little, at the bottom is an embedded Message Object. This will be used to set the value of member fields/variables etc to do some heavy lifting. You should notice the //load up the knowledge base, this is simply boiler plate code to load up the rules engine.

// go! is the interesting part that is happening in this example.
Step 1: is to create the new Message object and set the message to anything, in this case “Hello World”, and then set the Status to the Message.HELLO static final variable from the embedded Message class this will be used later to fire/evaluate the rules.

Step 2: insert the above created message into the knowledge session created above, and then call the fireAllRules method.

Now the rules engine takes over and looks for something that it can hit on. The first thing it will match in the DRL is the “Hello World” rule since it is looking to match the statusField to the Message.HELLO. That triggers the printing of the passed in message, and then sets the message to a new message, and then sets the status to Message.GOODBYE, this then triggers the GoodBye Rule and the printing of the goodbye message. This chaining of rules is not always recommended as a best practice, and of course it would be easy to create a loop.

If you want to update the “Hello World” message as shown:

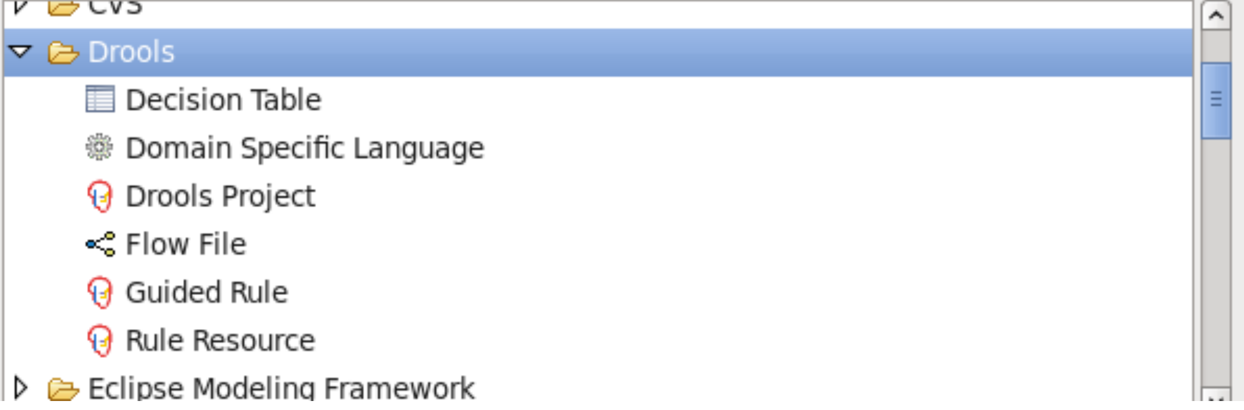
```
public static final void main(String[] args) {
    try {
        // load up the knowledge base
        KnowledgeBase kbase = readKnowledgeBase();
        StatefulKnowledgeSession ksession = kbase.newStatefulKn
        KnowledgeRuntimeLogger logger = KnowledgeRuntimeLoggerF
        // go !
        Message message = new Message();
        message.setMessage("Hello World, this is it");
        message.setStatus(Message.HELLO);
```

in the DroolsTest.java feel free, make sure you save it off, and rerun it to see how this has effected your test and output. Feel free to play around with this a little bit.

Lab Number 6: Custom Rules with Complex Event Processing

Lets create a simple DRL file that will trigger on a temperatures over 101.0 degrees fahrenheit. Imagine this is a simple check for wellness. We will add a few things to this over a few steps to explore the power of Complex Event Processing (CEP).

Right click on the src/main/rules and select New -> Other This will pop up the New icon, and open the Drools choices as shown:



Select Rule Resource, create a Temp.drl as shown:

Click “Finish”

Temp.drl should have these contents in it, make sure you do not replace the whole thing, but just the sections around rule 1 and 2.

```
rule "To High"

    when
        t: Temperature (value >= 101)
    then
       System.out.println("Setting too hot message");
       t.setMessage("Check the temperature too hot");
end

rule "To Low"

    when
        t: Temperature (value <= 96)
    then
         System.out.println("Setting too cold message");
       t.setMessage("Check the temperature too cold");
end
```

You will notice that our Temperature can not be resolved, lets fix that. Lets create a new Temperature object, Right click on com.sample and select Class, fill it in as shown:

Source folder: FirstOne/src/main/java Browse...
Package: com.sample Browse...
Enclosing type: Browse...
Name: Temperature
Modifiers: public default private protected
abstract final static

Click on Finish.

Now add in two variables:

```
private double value;
private String message;
```

as shown:

Double click on the yellow triangles to the left, and select the generate getters and setters for both of them.

You will do this for value and message, and end up with a file as shown:

Sample.drl | Temp.drl | Temperature.java

```
package com.sample;

public class Temperature {
   private double value;
   private String message;
public void setValue(double value) {
      this.value = value;
}
public double getValue() {
      return value;
}
public void setMessage(String message) {
      this.message = message;
}
public String getMessage() {
      return message;
}

}
```

Now if you click over to the Temp.drl, you will see the error message are gone, you might need to force a save by deleting a character and resaving. Please raise your hand if you can not get these errors to go away.

Now it is time to build a class that can execute these two created artifacts.

Right click on com.sample and select New->Class and fill it in as FirstTest and select the public static void main(S-tring args[]), as shown:

Source folder: FirstOne/src/main/java Browse...
Package: com.sample Browse...
Enclosing type: Browse...
Name: FirstTest
Modifiers: public default private protected
abstract final static
Superclass: java.lang.Object Browse...
Interfaces: Add...
Remove
Which method stubs would you like to create?
public static void main(String[] args)

Click Finish.

You will need to add in the following for as imports:

```
import org.drools.KnowledgeBase;
import org.drools.KnowledgeBaseFactory;
import org.drools.builder.KnowledgeBuilder;
import org.drools.builder.KnowledgeBuilderError;
import org.drools.builder.KnowledgeBuilderErrors;
import org.drools.builder.KnowledgeBuilderFactory;
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;
```

You will need to create a method as the following:

```
	private static KnowledgeBase readKnowledgeBase() throws Exception {
		KnowledgeBuilder kbuilder =
KnowledgeBuilderFactory.newKnowledgeBuilder();
		kbuilder.add(ResourceFactory.newClassPathResource("Temp.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;
	}
```

You should save the file off and have no errors, you will see a yellow box that show you are not yet using this method.

Now add to the main method the following lines:

```
try {
			// load up the knowledge base
			KnowledgeBase kbase = readKnowledgeBase();
			StatefulKnowledgeSession ksession =
kbase.newStatefulKnowledgeSession();
			KnowledgeRuntimeLogger logger =
KnowledgeRuntimeLoggerFactory.newFileLogger(ksession, "test");
			// go !
			Temperature temperature = new Temperature();
			temperature.setValue(105);
			ksession.insert(temperature);
			ksession.fireAllRules();
			logger.close();
		} catch (Throwable t) {
			t.printStackTrace();
		}
```

Save the file off, make sure there are no errors.
Right click on it and click on Run->Java Application

You should see a message in the Console like this:

```
Problems  Tasks  Seam Components  Servers  Console
<terminated> FirstTest [Java Application] /home/jimtyrrell/JDKs/jdk1.6.0_24/

Setting too hot message
```

Change the temperature to 95, what happens?
You should see that the too cold message was set as shown:

```
Setting too hot message
```

Now set it to 98 degrees, what happens...

You wills see that the program ends without any errors, lets add a toString method to the Tempature.java object as shown:

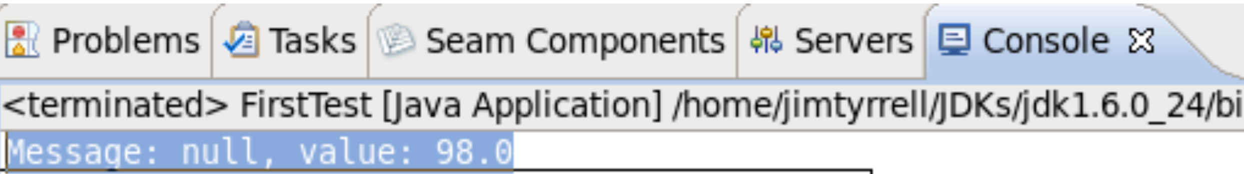
```
public String toString(){
        return "Message: " + message + ", value: " + value;
}
```

Make sure you save the file, now lets add our main method slightly, as shown:

```
// go !
Temperature temperature = new Temperature();
temperature.setValue(98);
ksession.insert(temperature);
ksession.fireAllRules();
logger.close();

System.out.println(temperature);
```

```
System.out.println(temperature);
```



Rerunning it, you should see the above output.

This code is not exactly any sort of complex event processing, in other words any time you see the value that is either too hot or too cold the rules engine will fire. We will need to add in a loop to trigger this over and over, however, that is not a good case as for every value too hot or cold the rules will fire, and this is not exactly what we want to do.

So lets make a few changes to our program, lets add in a loop. lets do 1000 iterations.

For the Temperature.java object add in the the following to the top of the program:

```
private int count = 0;

Temperature(int c)
{
        count = c;
}
```

```
public class Temperature {
    private double value;
    private String message;
    private int count = 0;

    Temperature(int c)
    {
        count = c;
    }
```

Update the toString method as shown:

```
return "Message: " + message + ", value: " + value + ", count: " + count;
```

Now lets open the FirstTest.java object and add in the following changes.

```
for(int c = 0; c < 1000; c++) M
```

Make sure you change the setValue method to 105 degrees

Pass in the c to the Temperature as shown:

Make it look as shown:

```
            for(int c = 0; c < 1000; c++)
            {
                Temperature temperature = new Temperature(c);
                temperature.setValue(105);
                ksession.insert(temperature);
                ksession.fireAllRules();
                logger.close();
            }
//          System.out.println(temperature);
```

Make sure you comment out the System.out.println method as shown.

Lets update the DRL to add in printing of the object information.

Run that and you will see thousands of lines of output in the console.

The DRL will now look like this:

```
rule "To High"

        when
                t: Temperature (value >= 101)
        then
           t.setMessage("Check the temperature too hot");
           System.out.println(t);
end

rule "To Low"

        when
                t: Temperature (value <= 96)
        then
           t.setMessage("Check the temperature too cold");
           System.out.println(t);
end
```

Now run the following and you should see 1000 lines of output, that will end like this:

```
<terminated> FirstTest [Java Application] /home/jimtyrrell/JDKs/jdk1.6.0_24/bin/ja
Message: Check the temperature too hot, value: 105.0, count: 992
Message: Check the temperature too hot, value: 105.0, count: 993
Message: Check the temperature too hot, value: 105.0, count: 994
Message: Check the temperature too hot, value: 105.0, count: 995
Message: Check the temperature too hot, value: 105.0, count: 996
Message: Check the temperature too hot, value: 105.0, count: 997
Message: Check the temperature too hot, value: 105.0, count: 998
Message: Check the temperature too hot, value: 105.0, count: 999
```

Not exactly Complex Event Processing, so lets add in a method that will randomly change the temperature that is passed in.

Lets add in the following method into FirstTest.java

```
public static double randomMovement(double currentValue, double max-
Swing)
        {
                return currentValue + (maxSwing * Math.random() - maxSwing/
2);
        }
```

This will allow us to randomly change the temperature.

Now lets update the main method:
add this before the for loop.
double value = 98.6;

and change the setValue method to be like the following.
temperature.setValue(*randomMovement*(value, 1));
value = temperature.getValue();

It will look like this:

```
            double value = 98.6;
            for(int c = 0; c < 1000; c++)
            {
                Temperature temperature = new Temperature(c);
                temperature.setValue(randomMovement(value, 1));
                ksession.insert(temperature);
                ksession.fireAllRules();
                value = temperature.getValue();
                logger.close();
            }
//          System.out.println(temperature);
```

Run that again.

You will notice it goes a few, couple, or several hundred times before starting to output the messages. Not exactly what you are hoping for, but just wait.....

We have a little more work to do stay tuned, email me at jtyrrell@redhat.com for updates!!!!!!!!

Sorry