JBOSS DROOLS COOKBOOK

Hot Recipes for the Drools Platform

Drools

JAVA CODE GEEKS

Java Code Geeks

Java 2 Java Developers Resource Center

JBoss Drools Cookbook

Contents

Preface

Drools is a business rule management system (BRMS) with a forward and backward chaining inference based rules engine, more correctly known as a production rule system, using an enhanced implementation of the Rete algorithm.

Drools supports the JSR-94 standard for its business rule engine and enterprise framework for the construction, maintenance, and enforcement of business policies in an organization, application, or service. (Source: https://en.wikipedia.org/wiki/Drools)

In this book, we provide a series of tutorials on Drools examples and business rules for a shopping domain model.

About the Author

JCGs (Java Code Geeks) is an independent online community focused on creating the ultimate Java to Java developers resource center; targeted at the technical architect, technical team lead (senior developer), project manager and junior developers alike.

JCGs serve the Java, SOA, Agile and Telecom communities with daily news written by domain experts, articles, tutorials, reviews, announcements, code snippets and open source projects.

You can find them online at https://www.javacodegeeks.com/

Chapter 1

JBoss Drools Tutorial for Beginners

If you consider any business process, you will see that is composed of one or more rules. Each rule may be responsible for some task based on some condition. As the rules grow it becomes difficult to manage the rules and to maintain them. This is why we need some tool to manage these rule and Drools fits the bill. It is a Business Rule Management System (BRMS) and rules engine written in Java. A rule could be:

- If you have a couple there is a discount of 5%
- If you purchase stuff for value more than 3000 bucks, you will get a flat 2% discount
- If you are a first time user, you will get 10 bucks into your wallet

In order to run through the rules, all we have to do is apply Customer and Cart state on the rules. Rules are static but data is dynamic so the data drives the decision process. Let's get started with setup and then examples.

This example uses the following frameworks:

- Maven 3.2.3
- Java 8
- Drools 6.2
- Eclipse as the IDE, version Luna 4.4.1.

1.1 Installation of Drools

Most simplest way to setup drools is to simply install Drools Eclipse Plugin.

- The rule workbench (for Eclipse) requires that you have Eclipse 3.4 or greater. It is also important that you install Eclipse GEF 3.4 or greater. You can install it using the built in update mechanism. Open the Help→Install New Software…→Add Site. Enter https://download.eclipse.org/tools/gef/updates/releases/ in Work with. Select GEF (Graphical Editing Framework). Press next, and agree to install the plug-in (an Eclipse restart may be required). Once this is completed, then you can continue on installing the rules plug-in.
- Download Drools Eclipse IDE Plugin. It is a zip file called `droolsjbpm-tools-distribution-6.2.0.Final` so you need to unzip the file and extract the contents. You need to install the plugin by opening Help→Install New Software…→Add Site. Enter the local location of `\droolsjbpm-tools-distribution-6.2.0.Final\binaries\org.drools.updatesite` , a name and click on Add. This will install the Drools Eclipse IDE plugin.
- Defining a Drools runtime - Download Drools Engine from Drools Download page. Its a zip file, since my drools ver# is 6.2, the file name is called `drools-distribution-6.2.0.Final.zip` . You need to unzip and extract contents. Go to Preferences→Drools→Installed Drools Runtime. Click on Add. Enter name and path of `drools-distribution-6.2.0.Final\binaries` . Click on OK to add the Drools Runtime Engine.

1.2 Create Drools Project

Click on File→New→Other→Drools→Drools Project to create a drools project based on the Drools Eclipse Plugin. The project will automatically create a sample drl file `Sample.drl` containing a rule. You will also find a java main class `DroolsTest` to test the rule.

`DroolsTest`:

```java
package com.sample;

import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

/**
 * 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
                KieServices ks = KieServices.Factory.get();
            KieContainer kContainer = ks.getKieClasspathContainer();
                KieSession kSession = kContainer.newKieSession("ksession-rules");

            // go !
            Message message = new Message();
            message.setMessage("Hello World");
            message.setStatus(Message.HELLO);
            kSession.insert(message);
            kSession.fireAllRules();
        } catch (Throwable t) {
            t.printStackTrace();
        }
    }

    public static class Message {

        public static final int HELLO = 0;
        public static final int GOODBYE = 1;

        private String message;

        private int status;

        public String getMessage() {
            return this.message;
        }

        public void setMessage(String message) {
            this.message = message;
        }

        public int getStatus() {
            return this.status;
        }

        public void setStatus(int status) {
            this.status = status;
        }
    }

}
```

`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( "Goodbye cruel world" );
        m.setStatus( Message.GOODBYE );
        update( m );
end

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

`Output:`

```
Hello World
Goodbye cruel world
```

1.3 Maven Dependencies

You can very well create a Maven project and specify the drools dependencies in `pom.xml` . If just want to rely on Maven, you need to add the below dependencies:

- `knowledge-api` - this provides the interfaces and factories
- `drools-core` - this is the core engine, runtime component. This is the only runtime dependency if you are pre-compiling rules.
- `drools-complier` - this contains the compiler/builder components to take rule source, and build executable rule bases. You don’t need this during runtime, if your rules are pre-compiled.

You just need to add `drools-compiler` dependency to your `pom.xml` , the other two dependencies are transitive dependencies.

`pom.xml:`

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/ ↩
    XMLSchema-instance"
        xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd ↩
            /maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.javacodegeeks.drools</groupId>
        <artifactId>droolsHelloWorld</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <dependencies>
                <dependency>
                        <groupId>org.drools</groupId>
                        <artifactId>drools-compiler</artifactId>
                        <version>${drools.version}</version>
                </dependency>
        </dependencies>
        <properties>
                <drools.version>6.2.0.Final</drools.version>
                <jbpm.version>6.2.0.Final</jbpm.version>
        </properties>
</project>
```

1.4 Define a Drools Rule

Here is a simple POJO which contains a topic and method called `introduceYourself()` .

`DroolsIntroduction:`

```
package com.javacodegeeks.drools;

/**
 * This is a sample class to launch a rule.
 */
public class DroolsIntroduction {
    private String topic;

    public DroolsIntroduction(String topic) {
        this.topic = topic;
    }

    public String getTopic() {
        return topic;
    }

    public String introduceYourself() {
        return "Drools 6.2.0.Final";
    }
}
```

Now let's define a simple rule. Rule is the above POJO will introduce itself only when the topic is about Drools.

Here is the rule:

`hello.drl:`

```
package com.javacodegeeks.drools;
rule "Drools Introduction"
when
$droolsIntro : DroolsIntroduction( topic == "Drools" )
then
System.out.println($droolsIntro.introduceYourself());
end
```

1.5 Rule Structure

You need to specify the package of the rule just the way we do it for java. Next follows the import statements, as seen here at the top of the file, are structured the same way as they are in Java class files. Each rule starts with `rule` and then its name. Next comes the condition `when ... then ...` end. To access the POJO property, you need to use it directly. You don't need to call the getters to access properties.

```
when $droolsIntro : DroolsIntroduction( topic == "Drools" )
```

The condition starts with the class name `DroolsIntroduction`. It checks whether the `topic` property is equal to *Drools*. If yes, we set the instance to a variable, `$droolsIntro`. Next, we take some action desired when our condition is met. We use the variable to call an instance method `$droolsIntro.introduceYourself()` and print the returned value using `System.out.println`.

```
then
System.out.println($droolsIntro.introduceYourself());
end
```

1.6 KIE Module

Now its the time to run the rule. In order to run the rule, Drools provides a configuration file called `kmodule.xml`. It acts as a descriptor that selects resources to knowledge bases and configures those knowledge bases and sessions.

`KBase` is a repository of all the application's knowledge definitions. Sessions are created from it and then data is inserted into the session which in turn will be used to start the process. Here is the configuration file which is located in `META-INF` directory.

`kmodule.xml:`

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns="https://jboss.org/kie/6.0.0/kmodule">
    <kbase name="rules" packages="rules">
        <ksession name="ksession-rules"/>
    </kbase>
</kmodule>
```

In order to run the rule, we need to add the below highlighted code. If yuou note, we create an instance of `DroolsIntroduction` and insert the instance to the session. Next, we call `kSession.fireAllRules()` run the rule.

`DroolsIntroduction:`

```
package com.javacodegeeks.drools;

import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

/**
 * This is a sample class to launch a rule.
 */
public class DroolsIntroduction {
    private String topic;

    public DroolsIntroduction(String topic) {
        this.topic = topic;
    }

    public static final void main(String[] args) {
        try {
            // load up the knowledge base
            KieServices ks = KieServices.Factory.get();
            KieContainer kContainer = ks.getKieClasspathContainer();
            KieSession kSession = kContainer.newKieSession("ksession-rules");

            DroolsIntroduction droolsIntroduction = new DroolsIntroduction("Drools");
            kSession.insert(droolsIntroduction);
            kSession.insert(new DroolsIntroduction("spring"));
            kSession.insert(new DroolsIntroduction("Drools"));
            kSession.fireAllRules();
        } catch (Throwable t) {
            t.printStackTrace();
        }
    }

    public String getTopic() {
        return topic;
    }

    public String introduceYourself() {
        return "Drools 6.2.0.Final";
    }
}
```

Only if the topic is *Drools*, you will see an introduction output. Since we have added `DroolsIntroduction` twice, we see output *Drools 6.2.0.Final* repeating.

`Output:`

```
Drools 6.2.0.Final
Drools 6.2.0.Final
```

1.7 Global Variable

In this example, I will show you how to assign a global variable to a session.

First declare the global in the rule file, `global String topicLevel`.

Next, set the global variable to the rule session.

`hello.drl:`

```
package com.javacodegeeks.drools;
global String topicLevel
rule "Drools Introduction"
when
$droolsIntro : DroolsIntroduction( topic == "Drools" )
then
System.out.println($droolsIntro.introduceYourself() + ", topic level is " + topicLevel);
end
```

`DroolsIntroduction:`

```
package com.javacodegeeks.drools;

import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

/**
 * This is a sample class to launch a rule.
 */
public class DroolsIntroduction {
    private String topic;

    public DroolsIntroduction(String topic) {
        this.topic = topic;
    }

    public static final void main(String[] args) {
        try {
            // load up the knowledge base
            KieServices ks = KieServices.Factory.get();
            KieContainer kContainer = ks.getKieClasspathContainer();
            KieSession kSession = kContainer.newKieSession("ksession-rules");

            DroolsIntroduction droolsIntroduction = new DroolsIntroduction("Drools");
            kSession.insert(droolsIntroduction);
            kSession.insert(new DroolsIntroduction("spring"));
            kSession.insert(new DroolsIntroduction("Drools"));
            kSession.fireAllRules();

            kSession.setGlobal("topicLevel", "Beginner");
            kSession.insert(new DroolsIntroduction("Drools"));
            kSession.fireAllRules();
        } catch (Throwable t) {
            t.printStackTrace();
        }
    }

    public String getTopic() {
        return topic;
    }

    public String introduceYourself() {
        return "Drools 6.2.0.Final";
    }
}
```

`Output:`

```
Drools 6.2.0.Final, topic level is null
Drools 6.2.0.Final, topic level is null
Drools 6.2.0.Final, topic level is Beginner
```

1.8 Functions

To get rid of the *null* topic level, we will introduce a new function called `getDefaultIfNull()` to our rule.

It will return a default value of the level if the global variable is not set to the session.

`hello.drl:`

```
package com.javacodegeeks.drools;
global String topicLevel
```

```
rule "Drools Introduction"
when
$droolsIntro : DroolsIntroduction( topic == "Drools" )
then
System.out.println($droolsIntro.introduceYourself() + ", topic level is " + ↩
    getDefaultIfNull(topicLevel));
end
function String getDefaultIfNull(String topicLevel) {
return topicLevel == null ? "Moderate" : topicLevel;
}
```

If you run the rule now, you will get a default topic level in the output.

`Output:`

```
Drools 6.2.0.Final, topic level is Moderate
Drools 6.2.0.Final, topic level is Moderate
Drools 6.2.0.Final, topic level is Beginner
```

1.9 Download the Eclipse Project

This was a beginner tutorial on Drools.

Download

You can download the full source code of this example here: **DroolsBasicsMavenWay.zip**

Chapter 2

Drools Rule Engine Tutorial

Drools is a Rule Engine that uses the rule-based approach to decouple logic from the system. The logic is external to the system in form of rules which when applied to data results into the decision making. A rules engine is a tool for executing business rules. In this article, we will write some business rules for a shopping domain model.

If you want to more know about Drools Introduction or its setup, read here.

This example uses the following frameworks:

- Maven 3.2.3
- Java 8
- Drools 6.2
- Eclipse as the IDE, version Luna 4.4.1.

In your `pom.xml` , you need to add the below dependencies:

- `knowledge-api` - this provides the interfaces and factories
- `drools-core` - this is the core engine, runtime component. This is the only runtime dependency if you are pre-compiling rules.
- `drools-complier` - this contains the compiler/builder components to take rule source, and build executable rule bases. You don't need this during runtime, if your rules are pre-compiled.

2.1 Dependencies

pom.xml:

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/ ←
    XMLSchema-instance"
        xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd ←
            /maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.javacodegeeks.drools</groupId>
        <artifactId>droolsHelloWorld</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <dependencies>
                <dependency>
                        <groupId>org.drools</groupId>
                        <artifactId>drools-compiler</artifactId>
                        <version>${drools.version}</version>
                </dependency>
        </dependencies>
        <properties>
                <drools.version>6.2.0.Final</drools.version>
                <jbpm.version>6.2.0.Final</jbpm.version>
        </properties>
</project>
```

2.2 Shopping Cart Domain Model

Customer will add one or more products to the cart. There are certain rules that we want to fire as we process the cart. Rules are:

- If a product needs registration, customer need to register else the item will not be processed.
- There will be discounts applied to the cart total price. If the customer has just registered to the site, there will be a 2% discount on the first purchase.
- If the customer has a coupon, another 5% discount will be applied on the total price. The coupon code and the percentage amounts may vary.
- If customer's requested quantity of product exceeds the available stock, then it will be registered as an issue.
- If a product turns out-of-stock, an error will be registered.

2.3 Our First Drools Rule

Business rules are composed of facts and conditional statements.

Before we get to the structure of the rule, lets see our first rule. We will add more rules to the file as we progress further.

```
package com.javacodegeeks.drools;
import com.javacodegeeks.drools.Cart;
import com.javacodegeeks.drools.CartItem;
import com.javacodegeeks.drools.CartStatus;
import com.javacodegeeks.drools.Product;
import java.util.List;

global List outOfStockProducts;

function String pendingItemKey(CartItem cartItem) {
return cartItem.getCart().getCustomer().getId() + "-"+ cartItem.getProduct().getDesc();
}

//Is it out of stock?
rule "Is Out-Of Stock"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED && product.getAvailableQty() == 0)
then
System.out.println("\\nIs Out-Of Stock Rule");
System.out.println("************************************");
String error = "Can't process as " +  $cartItem.getProduct().getDesc() + " is Out-Of-Stock" ↩
     ;
System.out.println(error);
$cartItem.setErrors(true);
$cartItem.setError(error);
$cartItem.getCart().logItemError(pendingItemKey($cartItem), $cartItem);
outOfStockProducts.add($cartItem.getProduct());
end
```

Let’s examine the different parts of the rule in our next section.

2.4 Structure of a Rule File

- **package** - The first statement begins with the package name. A package is the name of the folder in which a rule file lives. This is useful for organizing our rules.

```
package com.javacodegeeks.drools;
```

- **imports** - We will specify the dependent fully classified java class names used in our rules.

```
import com.javacodegeeks.drools.Cart;
import com.javacodegeeks.drools.CartItem;
import com.javacodegeeks.drools.Product;
```

- **globals** - Using `global` we define global variables. We will use a global variable if we want it to be available for all the rules defined. It allow us to pass information into and out of our rules.

```
global List<Product> outOfStockProducts;
```

- **functions** - We will use a function if we want do some kind of processing on the data passed in and we need to do it multiple times in most of our rules. For example, below function manufactures a key using customer ID and product desc.

```
function String pendingItemKey(CartItem cartItem) {
return cartItem.getCart().getCustomer().getId() + "-"+ cartItem.getProduct().getDesc();
}
```

- **rules** - This is the *when then end* structure. In the when part we match a condition. If the condition holds true then the *then* part is executed.

If the cart item is still not processed, the rule checks whether the product is out of stock. If yes, then it logs an error, and moves the cart item to pending items.

```
rule "Is Out-Of Stock"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED && product.getAvailableQty() == 0)
then
System.out.println("\\nIs Out-Of Stock Rule");
System.out.println("************************************");
String error = "Can't process as " +  $cartItem.getProduct().getDesc() + " is Out-Of-Stock" ↩
     ;
System.out.println(error);
$cartItem.setErrors(true);
$cartItem.setError(error);
$cartItem.getCart().logItemError(pendingItemKey($cartItem), $cartItem);
outOfStockProducts.add($cartItem.getProduct());
end
```

2.5 Rule Name

The first part of the rule block starts with the rule name. For example:

```
rule "Is Out-Of Stock"
when
...
```

Next comes the *when* part where we add conditions on the Fact model

2.6 When Part of the Rule

Let’s look into the *when* part of the rule. We will go through the conditions that we have used in our rules.

- Match on cart. Since there is nothing in the bracket, it will match on whatever cart object is passed. It also assigns the fact model to a variable `$cart` .

```
when
$cart : Cart()
```

- If we don’t want to use the fact model in the then part, we can skip the variable assignment. It becomes too simple.

```
when
Cart()
```

- Calls to `cartItem.getProduct().getAvailableQty()` and verifies if the quantity is 0.

```
when
$cartItem : CartItem(product.getAvailableQty() == 0)
```

We want to make sure the rule fires only for *Not Yet* processed cart item so we will add an *&&* condition.

```
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED && product.getAvailableQty() == 0)
```

- Compares the quantity of the item to the product’s available quantity.

```
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, qty > product.getAvailableQty())
```

- Checks whether coupon code is equal to *DISC01*

```
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, cart.customer.coupon == 'DISC01')
```

- Boolean check - checks whether the customer is new

```
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, cart.customer.isNew())
```

- Multiple conditions - checks whether the product requires registration and whether customer has registered for the product.

```
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, product.isRequiresRegisteration(), ↩
      !cart.customer.isRegistered(product))
```

- Rule fires on a processed cart item.

```
when
$cartItem : CartItem(cartStatus == CartStatus.PROCESSED)
```

2.7 Then Part of the Rule

The *Then* side of a rule determines what will happen when there is at least one result in the *when* part of the rule.

In the *Then* part of the rule we can use anything that can be written in Java code.

For example:

```
rule "Is Out-Of Stock"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED && product.getAvailableQty() == 0)
then
System.out.println("\\nIs Out-Of Stock Rule");
System.out.println("************************************");
String error = "Can't process as " +  $cartItem.getProduct().getDesc() + " is Out-Of-Stock" ↩
     ;
System.out.println(error);
$cartItem.setErrors(true);
$cartItem.setError(error);
$cartItem.getCart().logItemError(pendingItemKey($cartItem), $cartItem);
outOfStockProducts.add($cartItem.getProduct());
end
```

As an alternative, the "then" part of a rule can be used to modify Working Memory. A common practice is to insert or update a fact into Working Memory when a rule is evaluated as true. We will see this in one of our next sections how the rules get re-evaluated.

2.8 Rule Attributes

Drools provides us with rule attribute to modify the behavior of a rule.

- `no-loop` - A rule may modify a fact in which case the rules will be re-evaluated. If a condition causes the same rule to fire again, it will end up modify the fact again, which will trigger the re-evaluation one more time. This may result into an infinite loop. Using `no-loop` , we can be assured that the rule cannot trigger itself.
- `salience` - It is used to set the priority of a rule. By default, all rules have a salience of zero, but can be given a positive or negative value. See salience example to know more about it.

```
rule "Print Cart Issues" salience -1
when
$cart : Cart()
then
if ($cart.hasIssues()) {
System.out.println("\\nPrint Cart Issues Rule");
System.out.println("************************************");
System.out.println($cart.getCartIssues());
insert($cart.getPendingItems());
}
end
```

- `dialect` - This specifies the syntax used in the rule. Currently, the options available are MVEL and Java.

2.9 Comments

These are pieces of text ignored by the rule engine. They can be on a single line (anything after // until the end of the line) or split over multiple lines (everything between the /* and */ Comments split over many lines). For example:

```
//Is it out of stock?
rule "Is Out-Of Stock"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED && product.getAvailableQty() == 0)
then
...
end
```

2.10 Working Memory

With Drools we have rules on one side and Working Memory on the other side. Application code will be responsible for loading appropriate facts into Working Memory and the rules will query on these facts to figure out whether to fire the rule or not. We don't have to load all the facts to the working memory and only the facts relevant to rules will be loaded. We can also load new fact or update an existing fact.

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

KieContainer kContainer = ks.getKieClasspathContainer();
KieSession kSession = kContainer.newKieSession("ksession-rules");
Customer newCustomer = Customer.newCustomer("JOHN01");
newCustomer.addItem(p1, 1);
newCustomer.addItem(p2, 2);
newCustomer.addItem(p4OutOfStock, 1);
newCustomer.addItem(p5, 10);

cartItems = newCustomer.getCart().getCartItems();
for (CartItem cartItem: cartItems) {
        kSession.insert(cartItem);
}
```

We will insert a new fact when we have come to a stage where some processing is done, a state change happened and there are rules which now fire on the new state.

2.11 Setting global variable

If you are using a global variable, you may have to set it on your working memory. It is a best practice to set all global values before asserting any fact to the working memory. For Example:

```
kSession.insert(newCustomer.getCart());
kSession.setGlobal("outOfStockProducts", new ArrayList());
```

Globals are not designed to share data between rules, if you want to pass data from rule to rule, we will have to load new facts into the working memory.

We will see in our next section how we can insert or update a fact.

2.12 Inserting New Fact

The *then* part of the rule can change the contents of the Working Memory. When this occurs, Drools will reevaluate all rules to see if any rules now evaluate to true. We will see two different ways by which we can change the working memory. Once the all the cart items are processed, we want to add the pending cart items to the working memory so that the rules on the pending items can fire.

```
package com.javacodegeeks.drools;
import com.javacodegeeks.drools.Cart;
import com.javacodegeeks.drools.CartItem;
import com.javacodegeeks.drools.CartStatus;

rule "Print Cart Issues" salience -1
when
$cart : Cart()
then
if ($cart.hasIssues()) {
System.out.println("\\nPrint Cart Issues Rule");
System.out.println("************************************");
System.out.println($cart.getCartIssues());
insert($cart.getPendingItems());
}
end
```

In the above rule, we have added pending items to the working memory. Once we have done it, its going to re-evaluate the rules so any rules on *PendingItems* will fire now.

```
Below rule simply prints the pending items.
rule "Print Pending Items"
when
$pendingItems : PendingItems()
then
System.out.println("\\nPrint Pending Items Rule");
System.out.println("************************************");
for (CartItem cartItem : $pendingItems.getCartItems()) {
System.out.println(cartItem);
}
end
```

2.13 Updating a Fact

The `update()` statement is similar to insert, but is used where the fact existed before the rule started.

For example:

```
rule "Mark the items processed" salience -2
when
$cart : Cart()
then
System.out.println("\\nMark the items processed Rule");
System.out.println("************************************");
for (CartItem cartItem : $cart.getCartItems()) {
```

```
if (cartItem.getCartStatus() != CartStatus.NEW || cartItem.getCartStatus() != CartStatus. ↩
    PENDING) {
cartItem.updateAsProcessed();
System.out.println(cartItem + " is processed");
update(cartItem);
}
}
end
```

2.14 More than on Rule

A rule file can have more than one rules. For example:

cartRules.drl:

```
package com.javacodegeeks.drools;
import com.javacodegeeks.drools.Cart;
import com.javacodegeeks.drools.CartItem;
import com.javacodegeeks.drools.CartStatus;

rule "Print Cart Issues" salience -1
when
$cart : Cart()
then
...
end

rule "Print Pending Items"
when
$pendingItems : PendingItems()
then
...
end

rule "Mark the items processed" salience -2
when
$cart : Cart()
then
...
end
```

cartItems.drl:

```
package com.javacodegeeks.drools;
import com.javacodegeeks.drools.Cart;
import com.javacodegeeks.drools.CartItem;
import com.javacodegeeks.drools.CartStatus;
import com.javacodegeeks.drools.Product;
import java.util.List;

global List<Product> outOfStockProducts;

function String pendingItemKey(CartItem cartItem) {
return cartItem.getCart().getCustomer().getId() + "-"+ cartItem.getProduct().getDesc();
}
//Is it out of stock?
rule "Is Out-Of Stock"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED && product.getAvailableQty() == 0)
then
```

```
...
end

rule "Verify Qty"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, qty > product.getAvailableQty())
then
...
end

rule "If has coupon, 5% discount"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, cart.customer.coupon == 'DISC01')
then
...
end

rule "If new, 2% discount"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, cart.customer.isNew())
then
...
end

rule "Has customer registered for the product?" salience 1
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, product.isRequiresRegisteration(), ↩
     !cart.customer.isRegistered(product))
then
...
end

rule "Add Processed CartItem to Order"
when
$cartItem : CartItem(cartStatus == CartStatus.PROCESSED)
then
...
end
```

You will see the entire rule contents when we run the example.

2.15 More than one Rule file

If your application ends up using large number of rules, you should be able to manage them by spreading them across files. For example, as you can see the section before, we have added rules to two different files `cartRules.drl` and `cartItemRules.drl`.

One file consists of cart item based rules and the other cart based rules.

2.16 Domain Model

Our application is about shopping cart. There are rules related to cart items and then cart. A cart item's initial status is set to NEW.

Once all cart item rules pass, the item is considered as processed and the status is updated to *PROCESSED*. Once an item is processed we update the cart item so that the rules can re-evaluated as we want rules on processed items to fire. If an item has one or more issues, its status is set to PENDING. We also have a rule on pending items.

Let’s go through the POJOs, rules and the code that fires the rules.

Product:

```java
package com.javacodegeeks.drools;

public class Product {
        private int price;
        private String desc;
        private int availableQty = 5;
        private boolean requiresRegistration;
        private boolean isOutOfStock;

        public void setRequiresRegistration(boolean requiresRegistration) {
                this.requiresRegistration = requiresRegistration;
        }

        public boolean isRequiresRegisteration() {
                return requiresRegistration;
        }

        public Product(String desc, int price) {
                this.desc = desc;
                this.price = price;
        }

        public int getPrice() {
                return price;
        }

        public String getDesc() {
                return desc;
        }

        public String toString() {
                return "product: " + desc + ", price: " + price;
        }

        public int getAvailableQty() {
                return availableQty;
        }

        public void setAvailableQty(int availableQty) {
                this.availableQty = availableQty;
        }

        public boolean isOutOfStock() {
                return isOutOfStock;
        }

        public void setOutOfStock(boolean isOutOfStock) {
                this.isOutOfStock = isOutOfStock;
        }

        public boolean equals(Object o) {
                if (o == null) {
                        return false;
                }
                if (!(o instanceof Product)) {
                        return false;
                }
                Product p = (Product) o;
                return getDesc().equals(p.getDesc());
        }

        public int hashCode() {
                return getDesc().hashCode();
        }
}
```

Customer:

```
package com.javacodegeeks.drools;

import java.util.ArrayList;
import java.util.List;


public class Customer {
        private String id;
        private Cart cart;
        private String coupon;
        private boolean isNew;
        private List<Product> registeredProducts = new ArrayList<Product>();

        public static Customer newCustomer(String id) {
                Customer customer = new Customer(id);
                customer.isNew = true;
                return customer;
        }

        private Customer(String id) {
                this.id = id;
        }

        public String getId() {
                return id;
        }

        public boolean isNew() {
                return isNew;
        }

        public void addItem(Product product, int qty) {
                if (cart == null) {
                        cart = new Cart(this);
                }
                cart.addItem(product, qty);
        }

        public String getCoupon() {
                return coupon;
        }

        public void setCoupon(String coupon) {
                this.coupon = coupon;
        }

        public Cart getCart() {
                return cart;
        }

        public void registerProduct(Product product) {
                registeredProducts.add(product);
        }
        public boolean isRegistered(Product p) {
                return registeredProducts.contains(p);
        }

        public String toString() {
                StringBuilder sb = new StringBuilder();
                sb.append("Customer new? ")
                   .append(isNew)
                   .append("\\nCoupon: ")
                   .append(coupon)
                   .append("\\n")
                   .append(cart);
                return sb.toString();
        }
}
```

Cart:

```
package com.javacodegeeks.drools;

import java.util.ArrayList;
import java.util.List;

public class Cart {
        private Customer customer;
        private List<CartItem> cartItems = new ArrayList<CartItem>();
        private double discount;
        private CartIssues cartIssues = new CartIssues();
        private PendingItems pendingItems = new PendingItems(customer);

        public Cart(Customer customer) {
                this.customer = customer;
        }

        public void addItem(Product p, int qty) {
                CartItem cartItem = new CartItem(this, p, qty);
                cartItems.add(cartItem);
        }

        public double getDiscount() {
                return discount;
        }

        public void addDiscount(double discount) {
                this.discount += discount;
        }

        public int getTotalPrice() {
                int total = 0;
                for (CartItem item : cartItems) {
                        if (item.hasErrors()) {
                                continue;
                        }
                        total += item.getProduct().getPrice() * item.getQty();
                }
                return total;
        }

        public Customer getCustomer() {
                return customer;
        }
        public List<CartItem> getCartItems() {
                return cartItems;
        }

        public void setCustomer(Customer customer) {
                this.customer = customer;
        }

        public int getFinalPrice() {
                return getTotalPrice() - (int) getDiscount();
        }

        public void logItemError(String key, CartItem cartItem) {
                cartIssues.logItemError(key, cartItem);
                pendingItems.addItem(cartItem);
                cartItem.setCartStatus(CartStatus.PENDING);
        }

        public String toString() {
                StringBuilder sb = new StringBuilder();
                for (CartItem cartItem : cartItems) {
                        sb.append(cartItem)
                          .append("\\n");
                }
                sb.append("Discount: ")
                  .append(getDiscount())
                  .append("\\nTotal: ")
                  .append(getTotalPrice())
                  .append("\\nTotal After Discount: ")
                  .append(getFinalPrice());
                return sb.toString();
        }

        public PendingItems getPendingItems() {
                return pendingItems;
        }

        public CartIssues getCartIssues() {
                return cartIssues;
        }

        public boolean hasIssues() {
                return cartIssues.hasIssues();
        }
}
```

CartItem:

```
package com.javacodegeeks.drools;

public class CartItem {
        private Cart cart;
        private Product product;
        private int qty;
        private boolean errors;
        private String error;
        private CartStatus cartStatus;

        public CartItem(Cart cart, Product product, int qty) {
                this.cart = cart;
                this.product = product;
                this.qty = qty;
                cartStatus = CartStatus.NEW;
        }

        public Product getProduct() {
                return product;
        }

        public int getQty() {
                return qty;
        }

        public String toString() {
                return "Product: " + product + ", qty: " + qty + ", processed: " + ↩
                    hasErrors() + (hasErrors() ? ", Issue: " + getError() : "");
        }

        public Cart getCart() {
                return cart;
        }

        public boolean hasErrors() {
                return errors;
        }

        public void setErrors(boolean errors) {
                this.errors = errors;
        }

        public String getError() {
                return error;
        }

        public void setError(String error) {
                this.error = error;
        }

        public void updateAsProcessed() {
                cartStatus = CartStatus.PROCESSED;
        }

        public CartStatus getCartStatus() {
                return cartStatus;
        }

        public void setCartStatus(CartStatus cartStatus) {
                this.cartStatus = cartStatus;
        }
}
```

CartStatus:

```
package com.javacodegeeks.drools;

public enum CartStatus {
NEW,
PROCESSED,
PENDING
}
```

CartIssues:

```
package com.javacodegeeks.drools;

import java.util.HashMap;
import java.util.Map;

public class CartIssues {
        private Map<String, CartItem> cartErrors = new HashMap<String, CartItem>();

        public void logItemError(String key, CartItem cartItem) {
                cartErrors.put(key,  cartItem);
        }

        public String toString() {
                StringBuilder sb = new StringBuilder();
                for (String key : cartErrors.keySet()) {
                        sb.append(key).append(cartErrors.get(key)).append("\\n");
                }
                return sb.toString();
        }

        public boolean hasIssues() {
                return !cartErrors.isEmpty();
        }
}
```

PendingItems:

```
package com.javacodegeeks.drools;

import java.util.ArrayList;
import java.util.List;

public class PendingItems {
        private Customer customer;
        private List<CartItem> cartItems = new ArrayList<CartItem>();

        public PendingItems(Customer customer) {
                this.customer = customer;
        }

        public Customer getCustomer() {
                return customer;
        }

        public List>CartItem< getCartItems() {
                return cartItems;
        }

        public void addItem(CartItem cartItem) {
                cartItems.add(cartItem);
        }
}
```

2.17 Rules

Most of the cart Item based rules fire for *Not yet* PROCESSED items. Some rules are there to calculate the discount of any. The other rules check whether the quantity is valid and whether the product is available.

If there are any issues, the items are moves to pending zone to be dealt later. You can think of a scenario where the customer is being notified as soon as the issues are resolved.

The rules are grouped by cart and cart items.

Once all the items are processed, the rules related to PROCESSED items will fire. Also, rules related to PENDING items will fire.

cartItems.drl:

```
package com.javacodegeeks.drools;
import com.javacodegeeks.drools.Cart;
import com.javacodegeeks.drools.CartItem;
import com.javacodegeeks.drools.CartStatus;
import com.javacodegeeks.drools.Product;
import java.util.List;

global List<Product> outOfStockProducts;

function String pendingItemKey(CartItem cartItem) {
return cartItem.getCart().getCustomer().getId() + "-"+ cartItem.getProduct().getDesc();
}
//Is it out of stock?
rule "Is Out-Of Stock"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED && product.getAvailableQty() == 0)
then
System.out.println("\\nIs Out-Of Stock Rule");
System.out.println("************************************");
String error = "Can't process as " +  $cartItem.getProduct().getDesc() + " is Out-Of-Stock" ↩
     ;
System.out.println(error);
$cartItem.setErrors(true);
$cartItem.setError(error);
$cartItem.getCart().logItemError(pendingItemKey($cartItem), $cartItem);
outOfStockProducts.add($cartItem.getProduct());
end

rule "Verify Qty"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, qty > product.getAvailableQty())
then
System.out.println("\\nVerify Qty Rule");
System.out.println("************************************");
String error = "Can't process as only " +  $cartItem.getProduct().getAvailableQty() + " of  ↩
    "
+ $cartItem.getProduct().getDesc() + " are left whereas qty requested is " + $cartItem. ↩
    getQty();
System.out.println(error);
 $cartItem.setErrors(true);
$cartItem.setError(error);
$cartItem.getCart().logItemError(pendingItemKey($cartItem), $cartItem);
end

rule "If has coupon, 5% discount"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, cart.customer.coupon == 'DISC01')
then
if (!$cartItem.hasErrors()) {
    System.out.println("\\nIf has coupon, 5% discount Rule");
    System.out.println("************************************");
    double discount = ((double)$cartItem.getCart().getTotalPrice())*0.05d;
    System.out.println("Coupon Rule: Process " + $cartItem.getProduct() + ", qty " +  ↩
        $cartItem.getQty() + ", apply discount " + discount);
    $cartItem.getCart().addDiscount(discount);
}
end

rule "If new, 2% discount"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, cart.customer.isNew())
then
if (!$cartItem.hasErrors()) {
    System.out.println("\\nIf new, 2% discount Rule");
    System.out.println("************************************");
    double discount = ((double)$cartItem.getCart().getTotalPrice())*0.2d;
    System.out.println("New Customer Rule: Process " + $cartItem.getProduct() + ", qty " + ↩
        $cartItem.getQty() + ", apply discount " + discount);
    $cartItem.getCart().addDiscount(discount);
}
end

rule "Has customer registered for the product?" salience 1
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, product.isRequiresRegisteration(), ↩
     !cart.customer.isRegistered(product))
then
System.out.println("\\nHas customer registered for the product? Rule");
System.out.println("************************************");
String error = "Can't process " + $cartItem.getProduct() + ", as requires registration. ↩
    Customer not registered for the product!";
System.out.println(error);
$cartItem.setErrors(true);
$cartItem.setError(error);
$cartItem.getCart().logItemError(pendingItemKey($cartItem), $cartItem);
end

rule "Add Processed CartItem to Order"
when
$cartItem : CartItem(cartStatus == CartStatus.PROCESSED)
then
System.out.println("\\nAdd Processed CartItem to Order Rule");
System.out.println("************************************");
System.out.println("Add to order " + $cartItem);
end
```

cart.drl:

```
package com.javacodegeeks.drools;
import com.javacodegeeks.drools.Cart;
import com.javacodegeeks.drools.CartItem;
import com.javacodegeeks.drools.CartStatus;

rule "Print Cart Issues" salience -1
when
$cart : Cart()
then
if ($cart.hasIssues()) {
System.out.println("\\nPrint Cart Issues Rule");
System.out.println("************************************");
System.out.println($cart.getCartIssues());
insert($cart.getPendingItems());
}
end

rule "Print Pending Items"
```

```
when
$pendingItems : PendingItems()
then
System.out.println("\\nPrint Pending Items Rule");
System.out.println("************************************");
for (CartItem cartItem : $pendingItems.getCartItems()) {
System.out.println(cartItem);
}
end

rule "Mark the items processed" salience -2
when
$cart : Cart()
then
System.out.println("\\nMark the items processed Rule");
System.out.println("************************************");
for (CartItem cartItem : $cart.getCartItems()) {
if (cartItem.getCartStatus() != CartStatus.NEW || cartItem.getCartStatus() != CartStatus. ↩
    PENDING) {
cartItem.updateAsProcessed();
System.out.println(cartItem + " is processed");
update(cartItem);
}
}
end
```

2.18 Let's fire the rules

We will create the session first. The rules are automatically read from the classpath and added to the session. Next, we will build the customer's cart and insert each cart item into the session. We will also insert cart as we do have rules that fire on cart fact.

DroolsRuleEngineExample:

```
package com.javacodegeeks.drools;

import java.util.ArrayList;
import java.util.List;

import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;


/**
 * This is a sample class to launch a rule.
 */
public class DroolsRuleEngineExample {

    public static final void main(String[] args) {
        try {
            // load up the knowledge base
            KieServices ks = KieServices.Factory.get();
            KieContainer kContainer = ks.getKieClasspathContainer();
            KieSession kSession = kContainer.newKieSession("ksession-rules");

            Customer customer = Customer.newCustomer("RS");
                Product p1 = new Product("Laptop", 15000);
                Product p2 = new Product("Mobile", 5000);
                p2.setRequiresRegistration(true);
                Product p3 = new Product("Books", 2000);
                Product p4OutOfStock = new Product("TV", 2000);
                p4OutOfStock.setAvailableQty(0);

                Product p5 = new Product("Tab", 10000);
                p5.setAvailableQty(2);

                customer.addItem(p1, 1);
                customer.addItem(p2, 2);
                customer.addItem(p3, 5);
                customer.setCoupon("DISC01");

                List<CartItem> cartItems = customer.getCart().getCartItems();
                for (CartItem cartItem: cartItems) {
                        kSession.insert(cartItem);
                }
                System.out.println("************* Fire Rules **************");
            kSession.fireAllRules();
            System.out.println("************************************");
            System.out.println("Customer cart\\n" + customer);

            Customer newCustomer = Customer.newCustomer("JOHN01");
                newCustomer.addItem(p1, 1);
                newCustomer.addItem(p2, 2);
                newCustomer.addItem(p4OutOfStock, 1);
                newCustomer.addItem(p5, 10);

                cartItems = newCustomer.getCart().getCartItems();
                for (CartItem cartItem: cartItems) {
                        kSession.insert(cartItem);
                }
                kSession.insert(newCustomer.getCart());
                kSession.setGlobal("outOfStockProducts", new ArrayList<Product>());
                System.out.println("************* Fire Rules **************");
            kSession.fireAllRules();
            System.out.println("************************************");
            System.out.println("Customer cart\\n" + customer);

        } catch (Throwable t) {
            t.printStackTrace();
        }
    }

}
```

Output:

```
************* Fire Rules **************

Has customer registered for the product? Rule
************************************
Can't process product: Mobile, price: 5000, as requires registration. Customer not  ↩
    registered for the product!

If has coupon, 5% discount Rule
************************************
Coupon Rule: Process product: Books, price: 2000, qty 5, apply discount 1250.0

If has coupon, 5% discount Rule
************************************
Coupon Rule: Process product: Laptop, price: 15000, qty 1, apply discount 1250.0
```

```
If new, 2% discount Rule
************************************
New Customer Rule: Process product: Books, price: 2000, qty 5, apply discount 5000.0

If new, 2% discount Rule
************************************
New Customer Rule: Process product: Laptop, price: 15000, qty 1, apply discount 5000.0
************************************
Customer cart
Customer new? true
Coupon: DISC01
Product: product: Laptop, price: 15000, qty: 1, processed: false
Product: product: Mobile, price: 5000, qty: 2, processed: true, Issue: Can't process  ←
    product: Mobile, price: 5000, as requires registration. Customer not registered for the  ←
    product!
Product: product: Books, price: 2000, qty: 5, processed: false
Discount: 12500.0
Total: 25000
Total After Discount: 12500
************* Fire Rules **************

Has customer registered for the product? Rule
************************************
Can't process product: Mobile, price: 5000, as requires registration. Customer not  ←
    registered for the product!

Is Out-Of Stock Rule
************************************
Can't process as TV is Out-Of-Stock

Verify Qty Rule
************************************
Can't process as only 2 of Tab are left whereas qty requested is 10

Verify Qty Rule
************************************
Can't process as only 0 of TV are left whereas qty requested is 1

If new, 2% discount Rule
************************************
New Customer Rule: Process product: Laptop, price: 15000, qty 1, apply discount 3000.0

Print Cart Issues Rule
************************************
JOHN01-TabProduct: product: Tab, price: 10000, qty: 10, processed: true, Issue: Can't  ←
    process as only 2 of Tab are left whereas qty requested is 10
JOHN01-MobileProduct: product: Mobile, price: 5000, qty: 2, processed: true, Issue: Can't  ←
    process product: Mobile, price: 5000, as requires registration. Customer not registered  ←
    for the product!
JOHN01-TVProduct: product: TV, price: 2000, qty: 1, processed: true, Issue: Can't process  ←
    as only 0 of TV are left whereas qty requested is 1


Print Pending Items Rule
************************************
Product: product: Mobile, price: 5000, qty: 2, processed: true, Issue: Can't process  ←
    product: Mobile, price: 5000, as requires registration. Customer not registered for the  ←
    product!
Product: product: TV, price: 2000, qty: 1, processed: true, Issue: Can't process as only 0  ←
    of TV are left whereas qty requested is 1
Product: product: Tab, price: 10000, qty: 10, processed: true, Issue: Can't process as only ←
     2 of Tab are left whereas qty requested is 10
```

```
Product: product: TV, price: 2000, qty: 1, processed: true, Issue: Can't process as only 0 ↩
    of TV are left whereas qty requested is 1

Mark the items processed Rule
************************************
Product: product: Laptop, price: 15000, qty: 1, processed: false is processed
Product: product: Mobile, price: 5000, qty: 2, processed: true, Issue: Can't process  ↩
    product: Mobile, price: 5000, as requires registration. Customer not registered for the  ↩
    product! is processed
Product: product: TV, price: 2000, qty: 1, processed: true, Issue: Can't process as only 0  ↩
    of TV are left whereas qty requested is 1 is processed
Product: product: Tab, price: 10000, qty: 10, processed: true, Issue: Can't process as only ↩
     2 of Tab are left whereas qty requested is 10 is processed

Add Processed CartItem to Order Rule
************************************
Add to order Product: product: Tab, price: 10000, qty: 10, processed: true, Issue: Can't  ↩
    process as only 2 of Tab are left whereas qty requested is 10

Add Processed CartItem to Order Rule
************************************
Add to order Product: product: TV, price: 2000, qty: 1, processed: true, Issue: Can't  ↩
    process as only 0 of TV are left whereas qty requested is 1

Add Processed CartItem to Order Rule
************************************
Add to order Product: product: Mobile, price: 5000, qty: 2, processed: true, Issue: Can't  ↩
    process product: Mobile, price: 5000, as requires registration. Customer not registered  ↩
    for the product!

Add Processed CartItem to Order Rule
************************************
Add to order Product: product: Laptop, price: 15000, qty: 1, processed: false
************************************
Customer cart
Customer new? true
Coupon: DISC01
Product: product: Laptop, price: 15000, qty: 1, processed: false
Product: product: Mobile, price: 5000, qty: 2, processed: true, Issue: Can't process  ↩
    product: Mobile, price: 5000, as requires registration. Customer not registered for the  ↩
    product!
Product: product: Books, price: 2000, qty: 5, processed: false
Discount: 12500.0
Total: 25000
Total After Discount: 12500
```

2.19 Download the Eclipse Project

This was a tutorial about JBoss Drools Rule Engine.

Download

You can download the full source code of this example here: **DroolsRuleEngineExamples.zip**

Chapter 3

JBoss Drools Guvnor Example

In this article, we will see an example of Drools Guvnor. We use Guvnor as *Business Rules Manager*. Guvnor is the name of the web and network related components for managing rules with drools.

If you want to more know about Drools Introduction or its setup, read here.

This example uses the following frameworks:

- Maven 3.2.3
- Guvnor 5.1.1

3.1 What is a Drools Guvnor?

In order to know Drools Guvnor, we first need to know what is a business rules manager. Business Rules manager allows people to manage rules in a multi user environment. It is a single point of truth for your business rules. It allows to manage the rules in a controlled fashion. We can keep track of different versions of the rules, its deployment. It also allows multiple users of different skill levels access to edit rules with user friendly interfaces. Guvnor is the name of the web and network related components for managing rules with drools.

3.2 Installing Guvnor

You should see the below login screen:

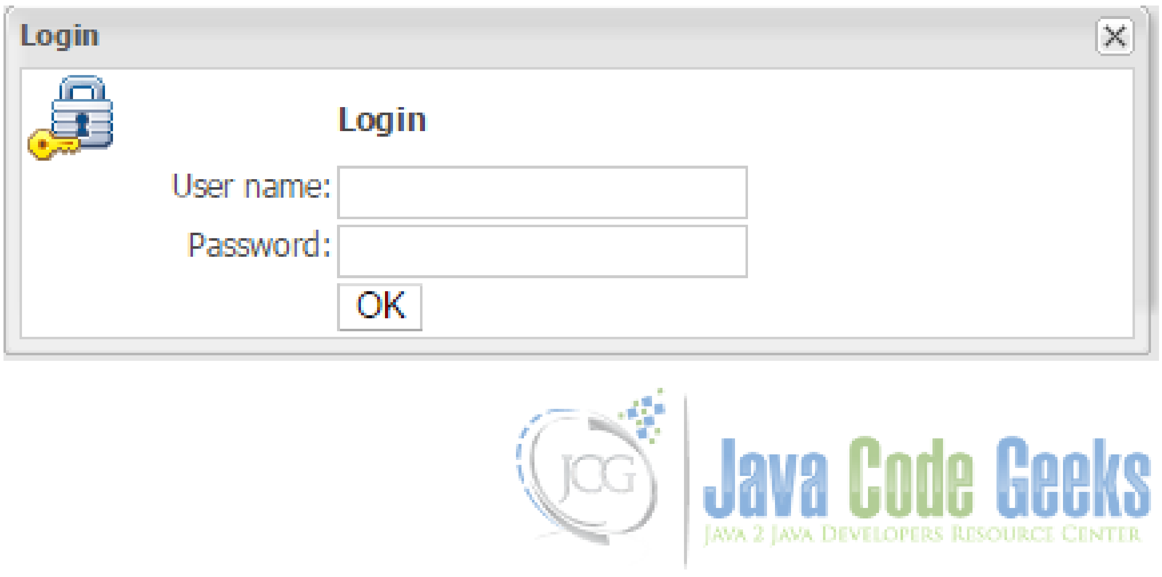
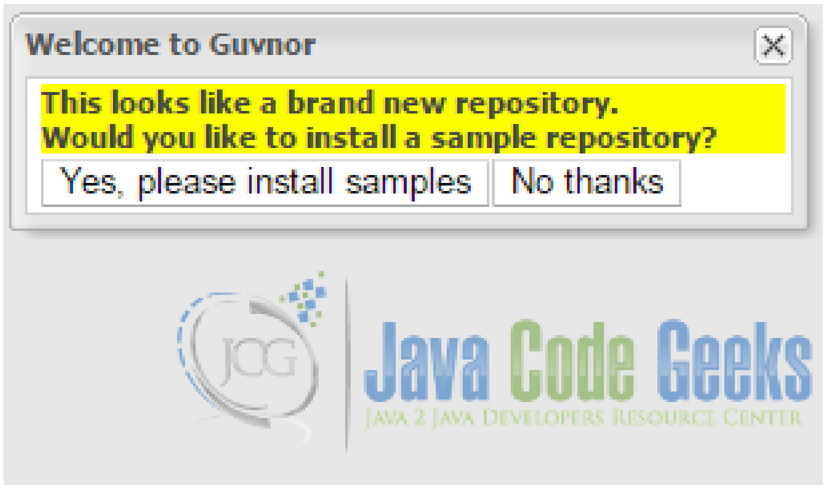


Figure 3.1: Guvnor Login

You don't need to enter user/password. Simply click on Ok. You will be taken to the welcome screen.


Welcome to Guvnor
Yes, please install samples
No thanks
Java Code Geeks
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

This looks like a brand new repository.
Would you like to install a sample repository?

Figure 3.2: Guvnor Welcome Screen

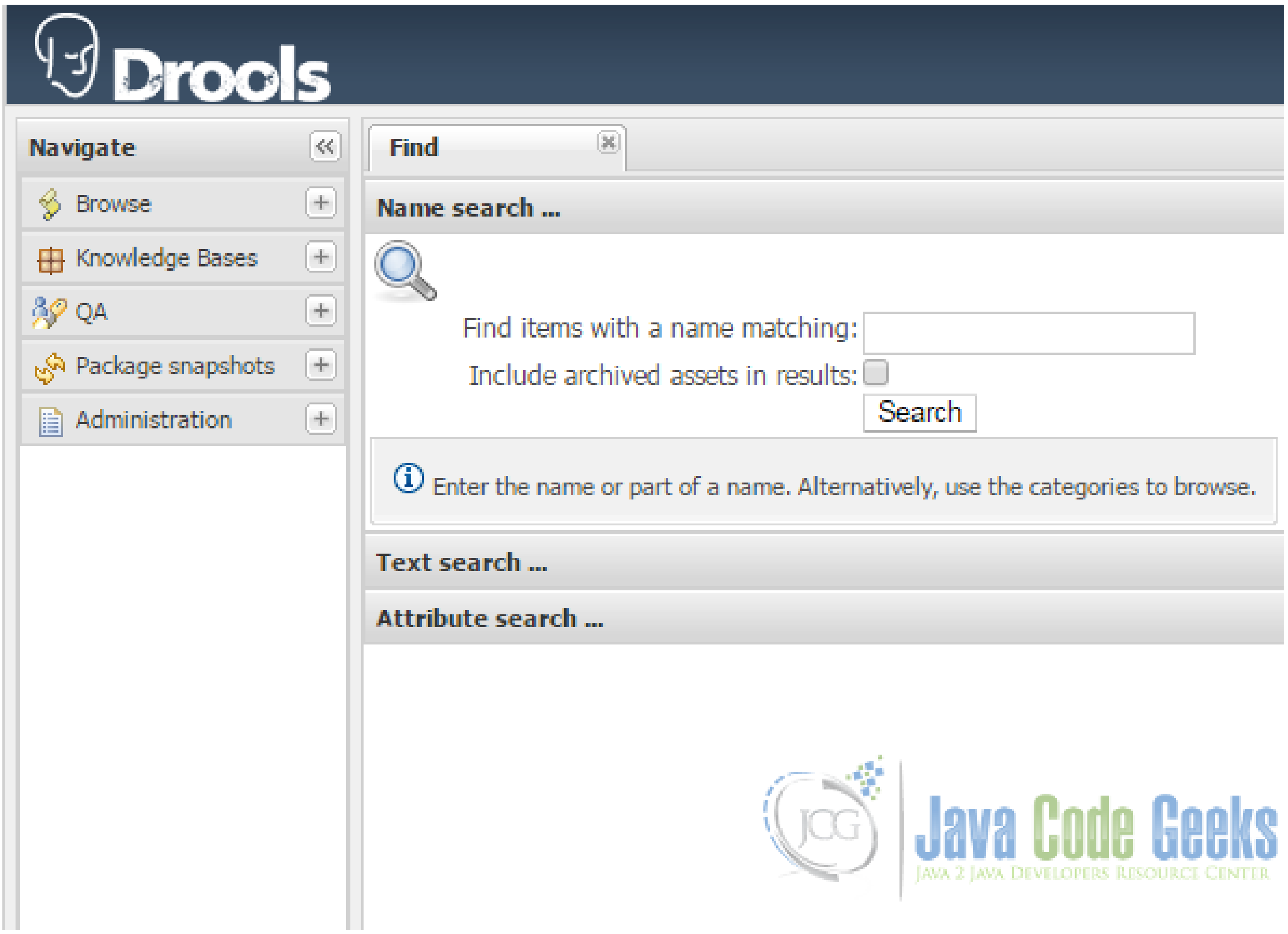
Click on *No thanks*.

3.3 Main Features of Guvnor

Browse the main areas. You will see:

- Knowledge Base
- QA
- Package Snapshots

- Administration



Find items with a name matching:
Include archived assets in results:

Enter the name or part of a name. Alternatively, use the categories to browse.

Figure 3.3: Main Features of Guvnor

3.4 Upload Domain Model to Guvnor

Before rules can be defined in Guvnor, a business model(a set of Java classes) jar needs to be uploaded to Guvnor. go to the "Knowledge Bases" tab. Click on "Create New" right under the tab and choose "New Package". Give your package a name - I choose "banking" - and press Ok. Then "Create new" - "Upload new Model Jar".

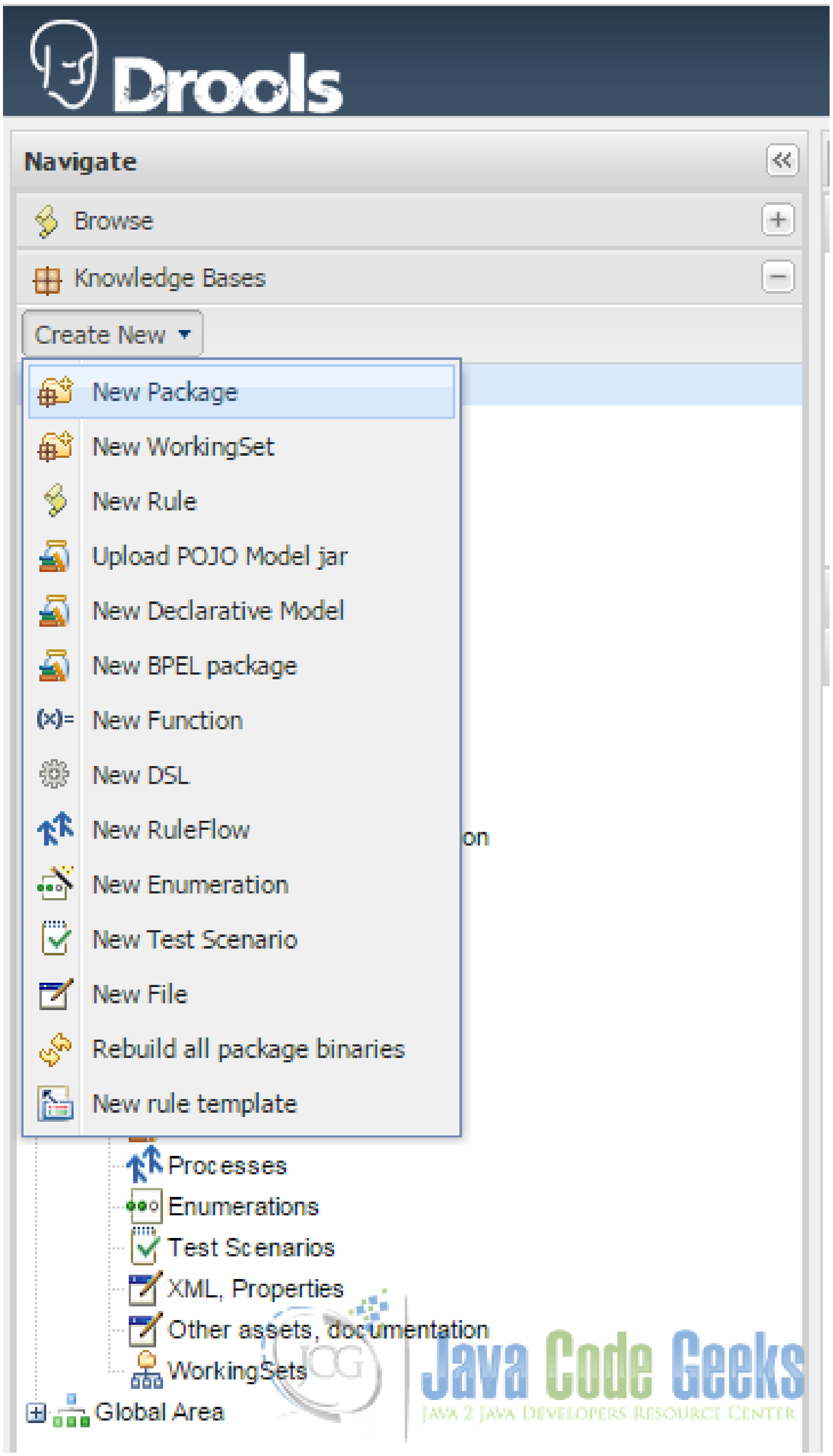
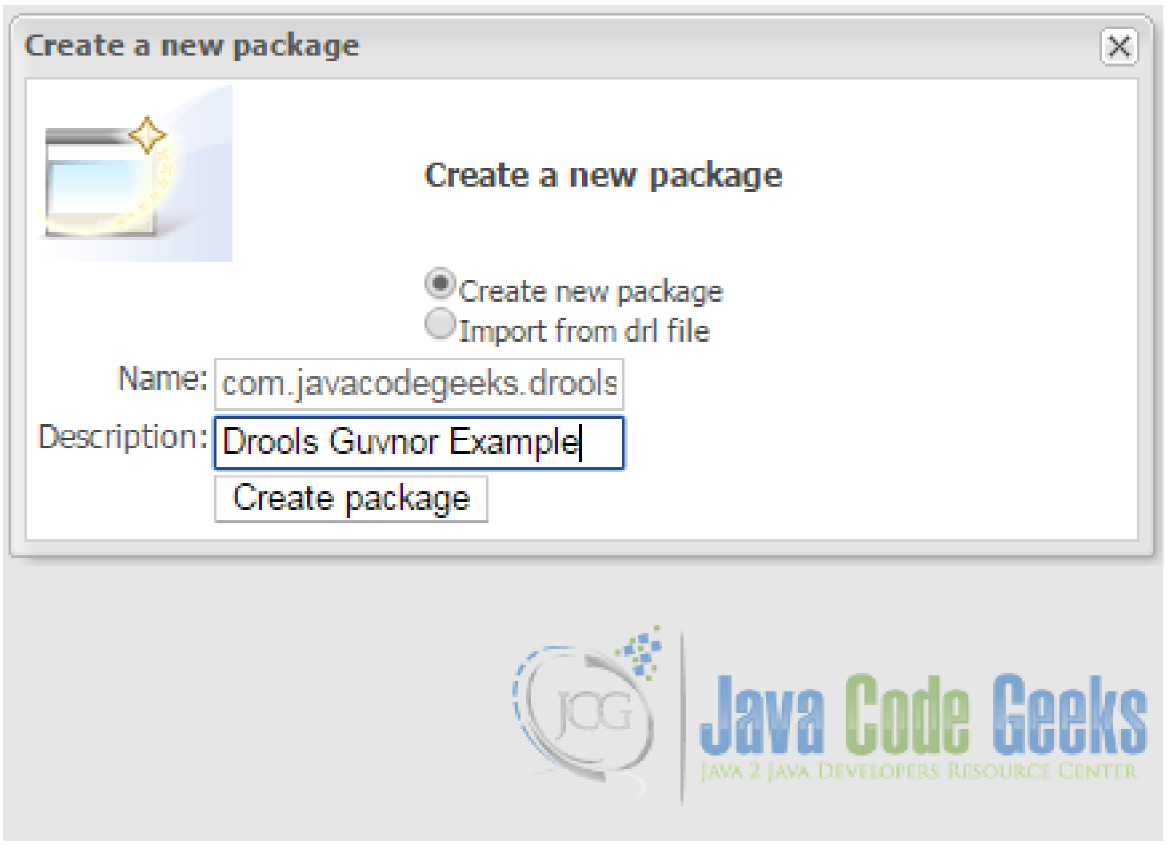


Figure 3.4: Create new Package

Create new package.

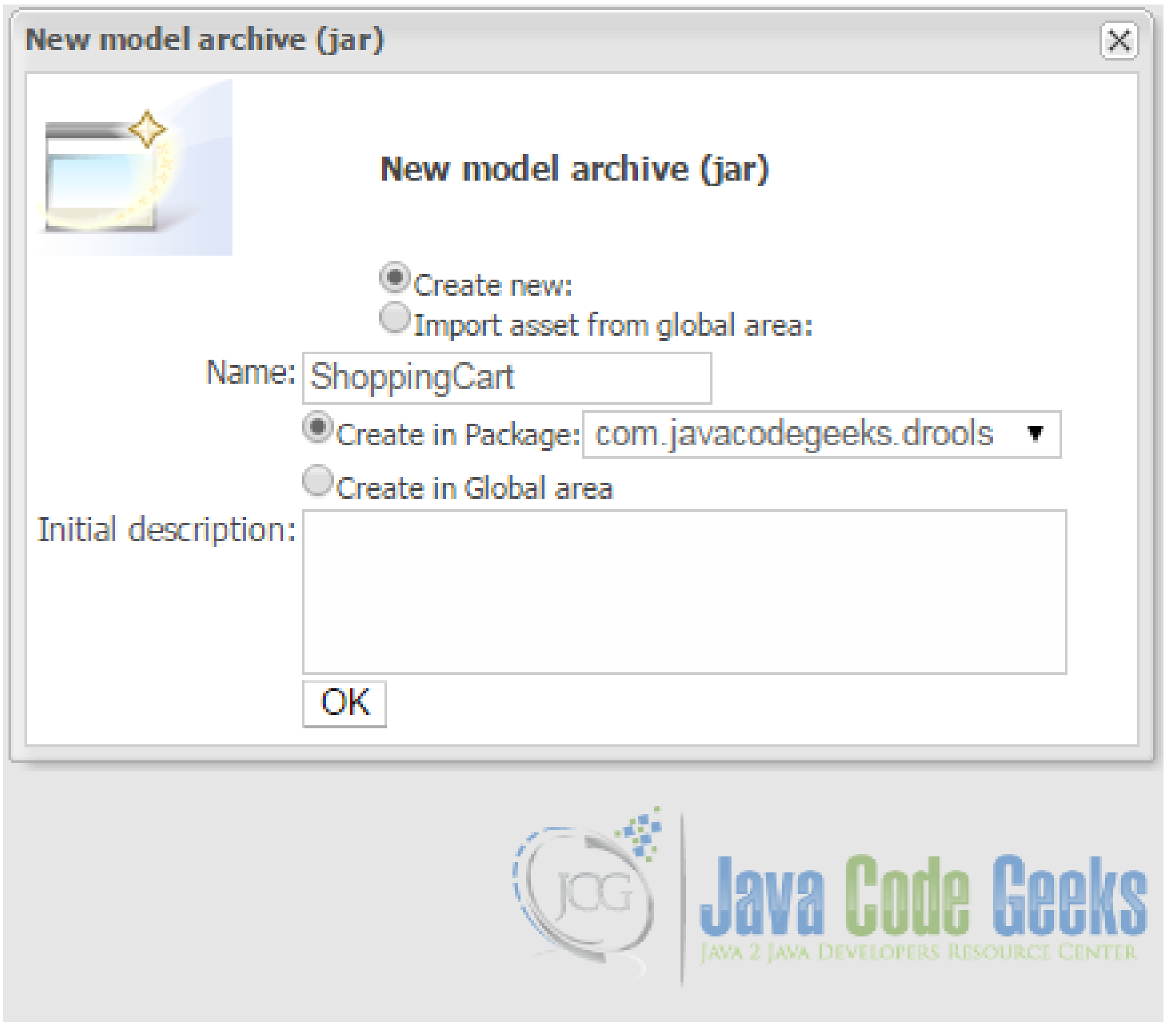


(●) Create new package
( ) Import from drl file

Name: com.javacodegeeks.drools
Description: Drools Guvnor Example

Figure 3.5: New package

Create new domain model.


New model archive (jar)

New model archive (jar)

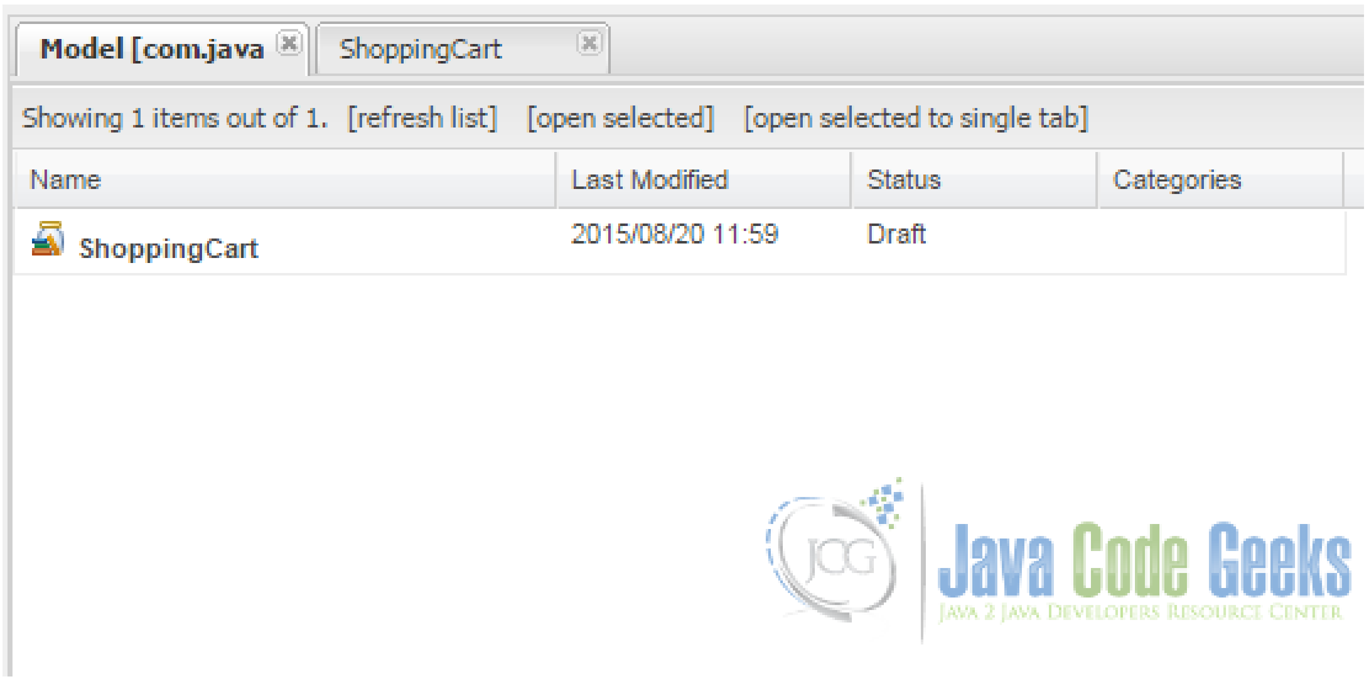
[✓ Create new:]
Import asset from global area:

Name: ShoppingCart
[✓ Create in Package:] com.javacodegeeks.drools
Create in Global area
Initial description:

OK

Figure 3.6: Create new model

You can see the new model in the *Model* tab.


Model [com.java
ShoppingCart
Showing 1 items out of 1. [refresh list] [open selected] [open selected to single tab]

Name	Last Modified	Status	Categories
ShoppingCart	2015/08/20 11:59	Draft	

Figure 3.7: Model Tab

Go to shopping car to upload the domain model.

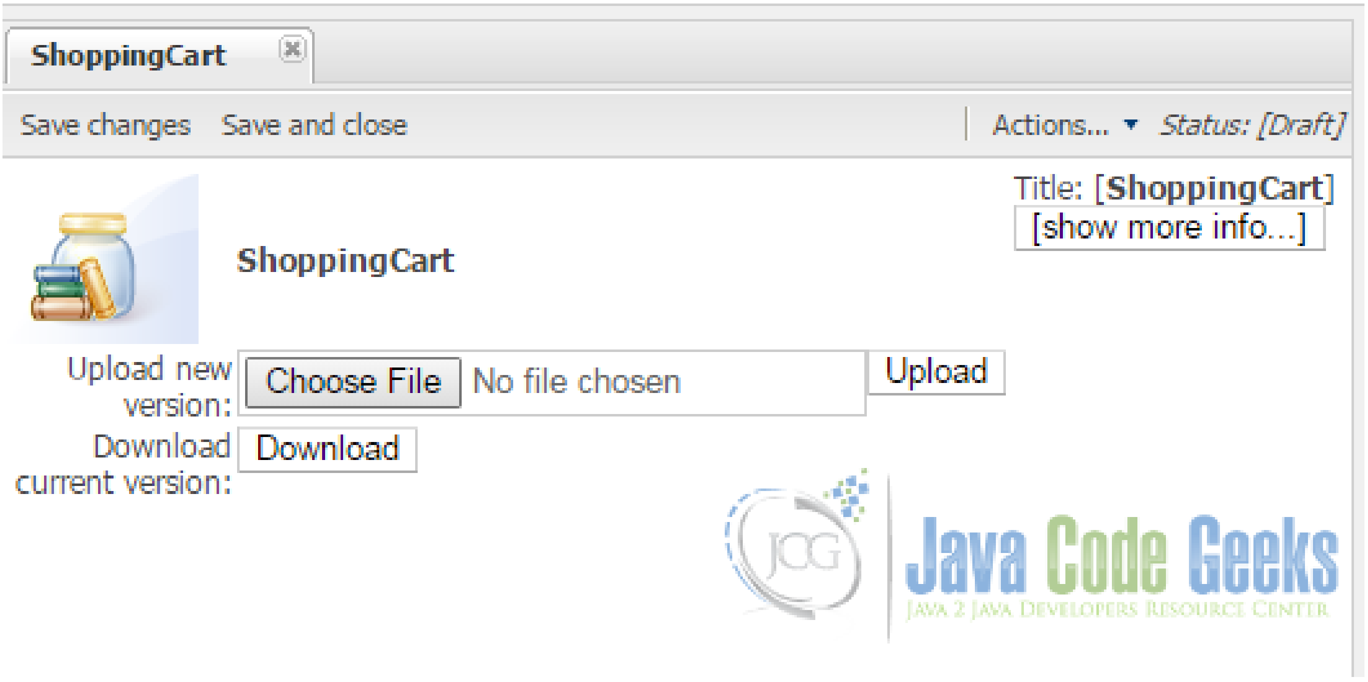


Figure 3.8: Upload domain model

Click on choose file to select the jar file.

You can either use maven to create jar file or use eclipse File→Export→Java→Jar file option to generate the domain model Jar file.

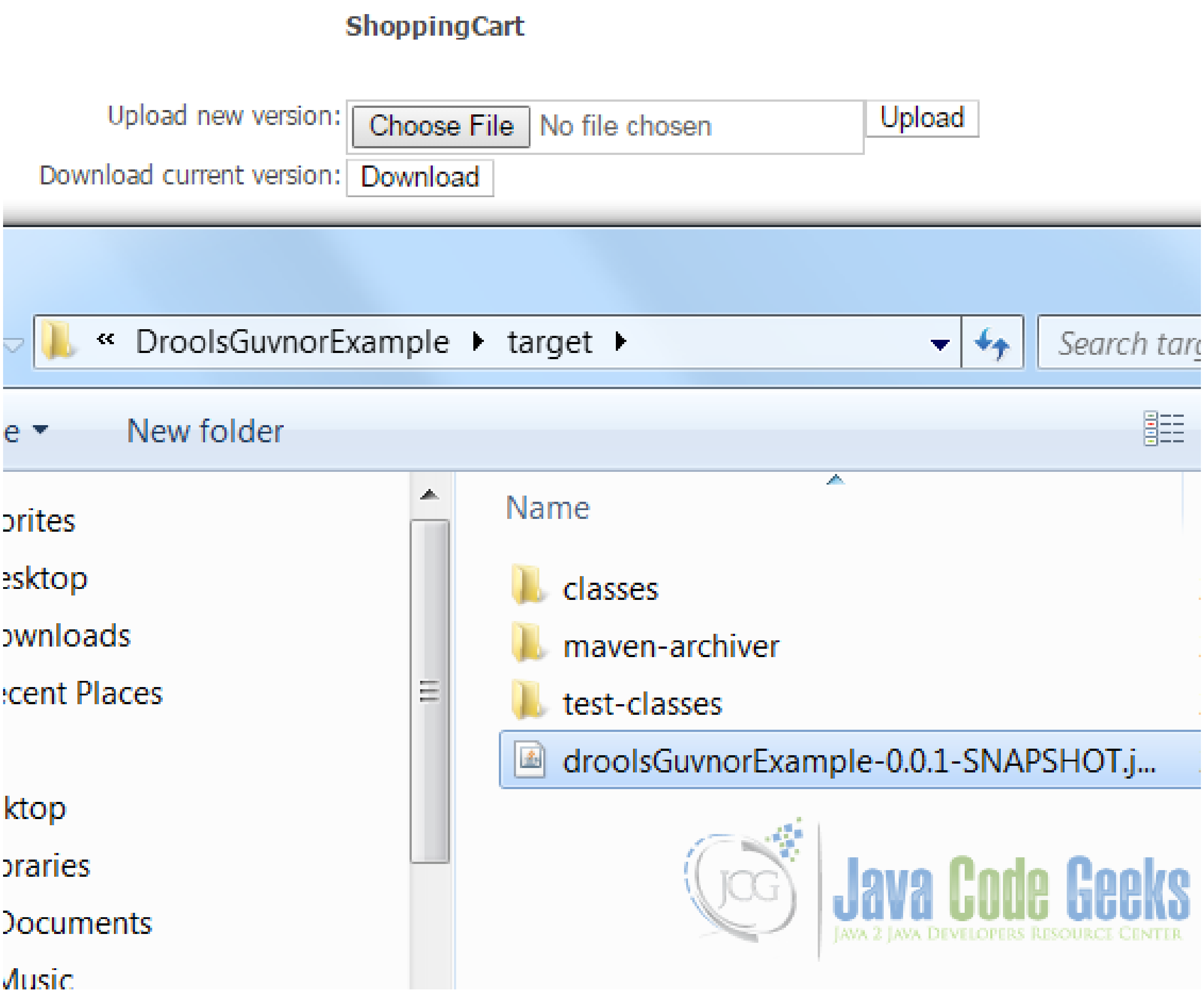
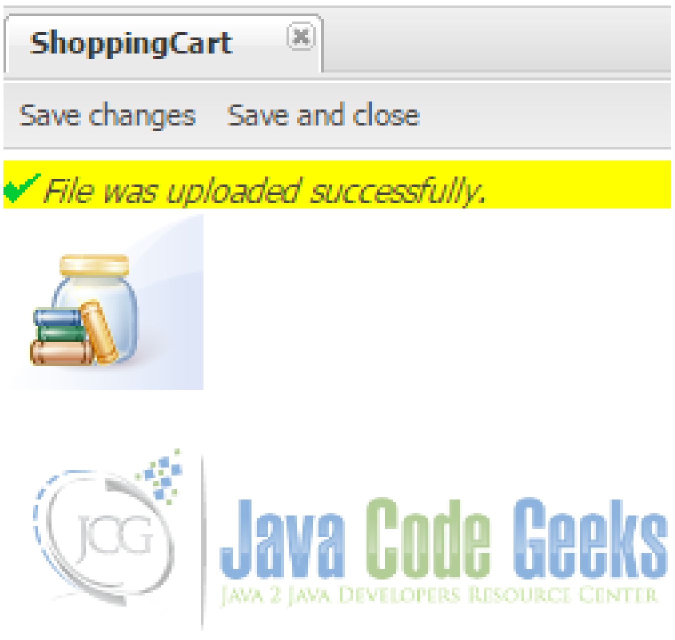


Figure 3.9: Select the domain model jar

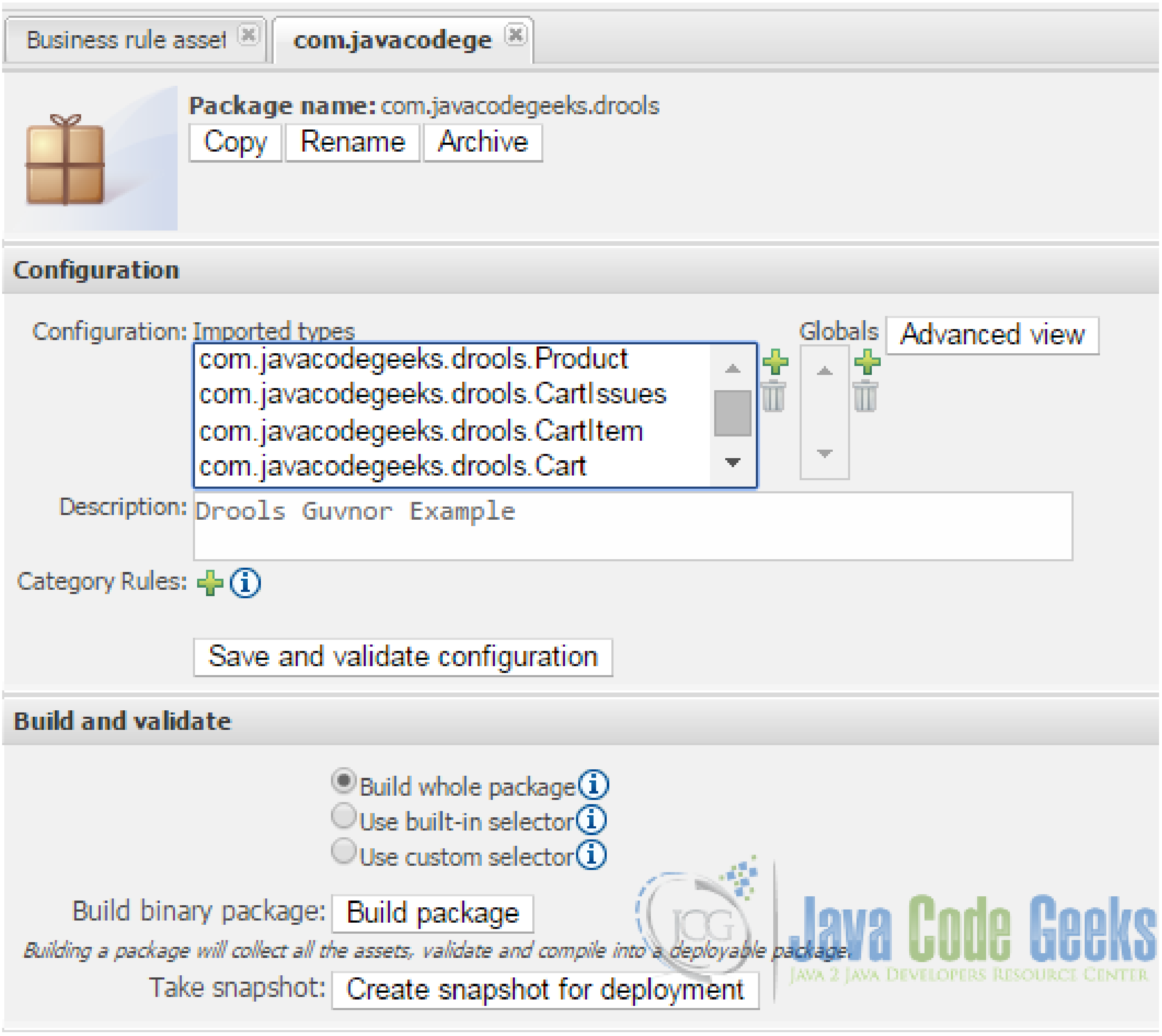
Once uploaded you will see a *Upload successful* message.



File was uploaded successfully.

Figure 3.10: Upload Successful Message

You can see the model classes in the package tab. Click on *Save and validate configuration* so that the model files become available as facts to the rules.


Business rule asset

com.javacodege

Package name: com.javacodegeeks.drools

Copy Rename Archive

Configuration

Configuration: Imported types

com.javacodegeeks.drools.Product
com.javacodegeeks.drools.CartIssues
com.javacodegeeks.drools.CartItem
com.javacodegeeks.drools.Cart

Globals Advanced view

Description: Drools Guvnor Example

Category Rules:

Save and validate configuration

Build and validate

[✓ Build whole package]
Use built-in selector
Use custom selector

Build binary package: Build package

Building a package will collect all the assets, validate and compile into a deployable package.

Take snapshot: Create snapshot for deployment

Figure 3.11: Package View

3.5 Define Rules

Adding a sample rule. Now the model is uploaded, we can start defining rules. To add a rule, you have to create a Category first (Administration>Category>New Category).

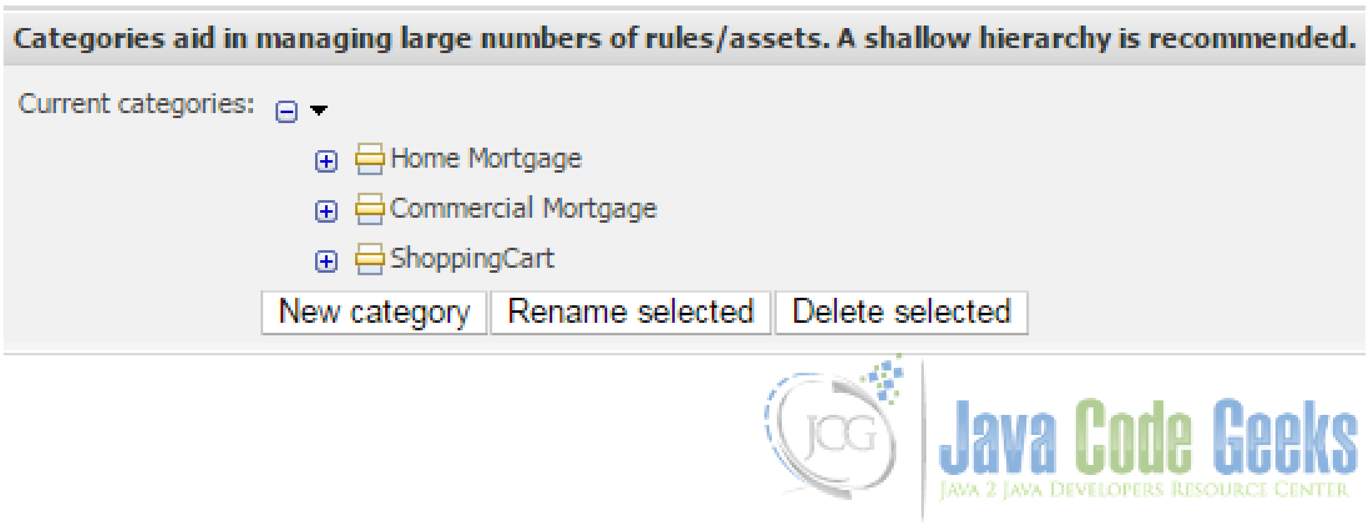
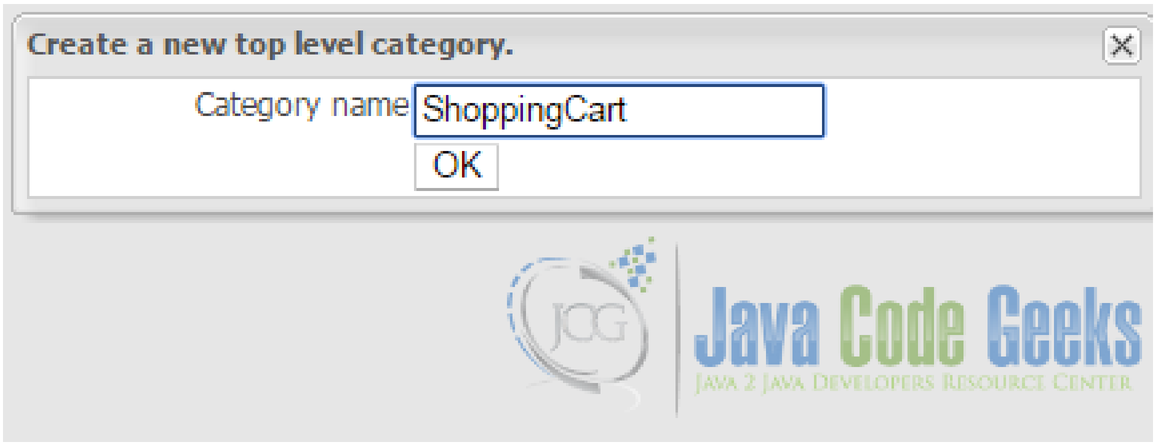


Figure 3.12: New Category

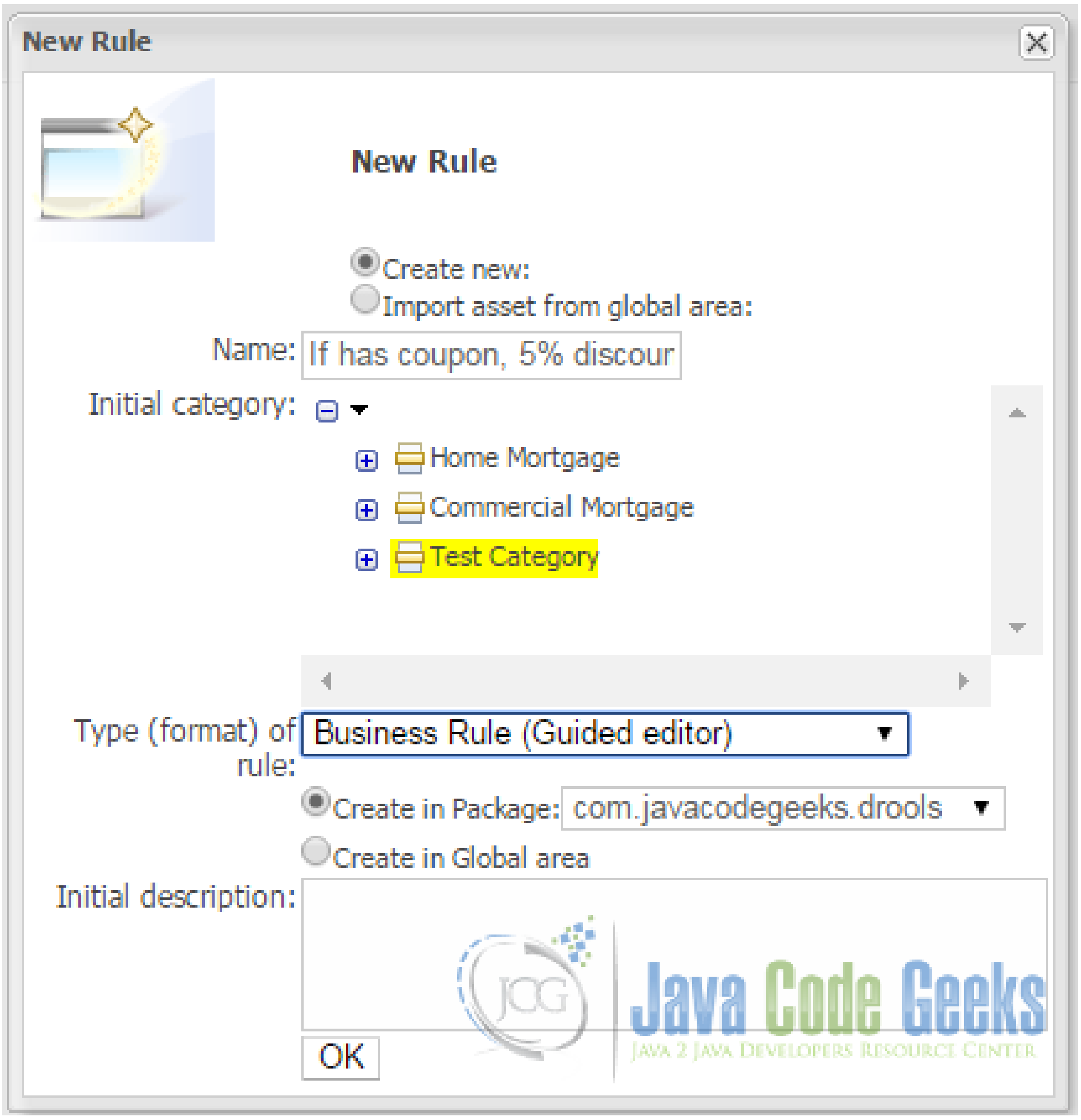
Create *ShoppingCart* category.



Category name: ShoppingCart

Figure 3.13: Create new category *ShoppingCart*

Once the category is created, will go back to the "Knowledge Bases" tab and choose Create New>New Rule to create our first rule.



Create new: [✓]
Import asset from global area:
Name: If has coupon, 5% discour
Initial category:
Type (format) of rule: Business Rule (Guided editor)
Create in Package: [✓] com.javacodegeeks.drools
Create in Global area
Initial description:

Figure 3.14: Create new rule

Click on the + green button to add conditions. To add a condition to the rule, we need to choose fact type as Cart. After choosing this, we'll automatically be taken back to the guided rule-editing screen. This is where we define the rule.

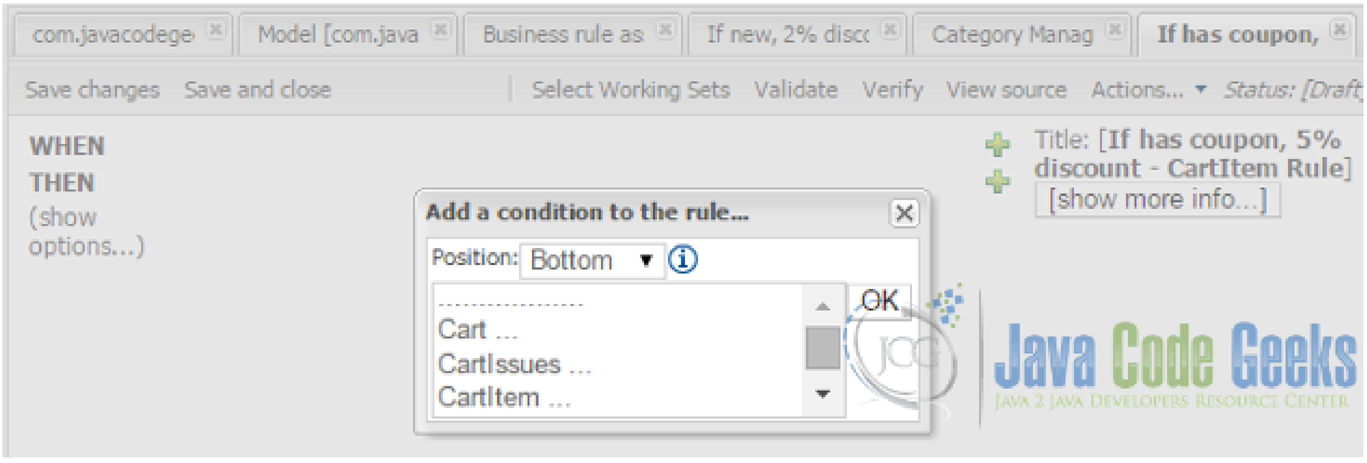
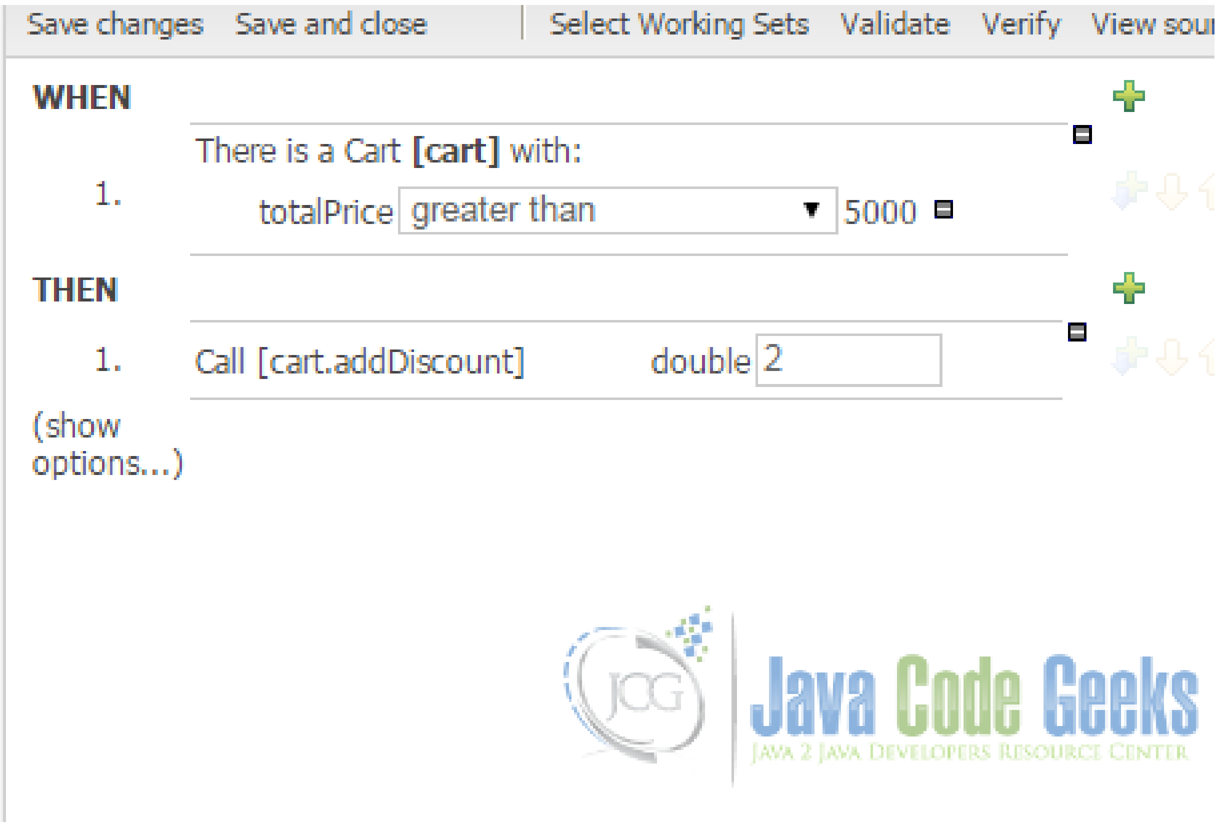


Figure 3.15: Add conditions to rule

3.6 Define the rule

We'll see that Cart has been added as a condition. Currently, this rule will fire for all carts, we need to add constraint that it fires only when the total price > 5000. To add constraint, we need to click on the + button on the WHEN condition. We'll choose total price as the field and *Greater than* as constraint with value set to 5000. Next, we will add *THEN* part. Click on the green plus sign next to the Then section. The Add a new action dialog box will be displayed. Here we will select *call on method* cart.addDiscount(), set the value to 2%.



WHEN

1. There is a Cart **[cart]** with:
   totalPrice greater than 5000

THEN

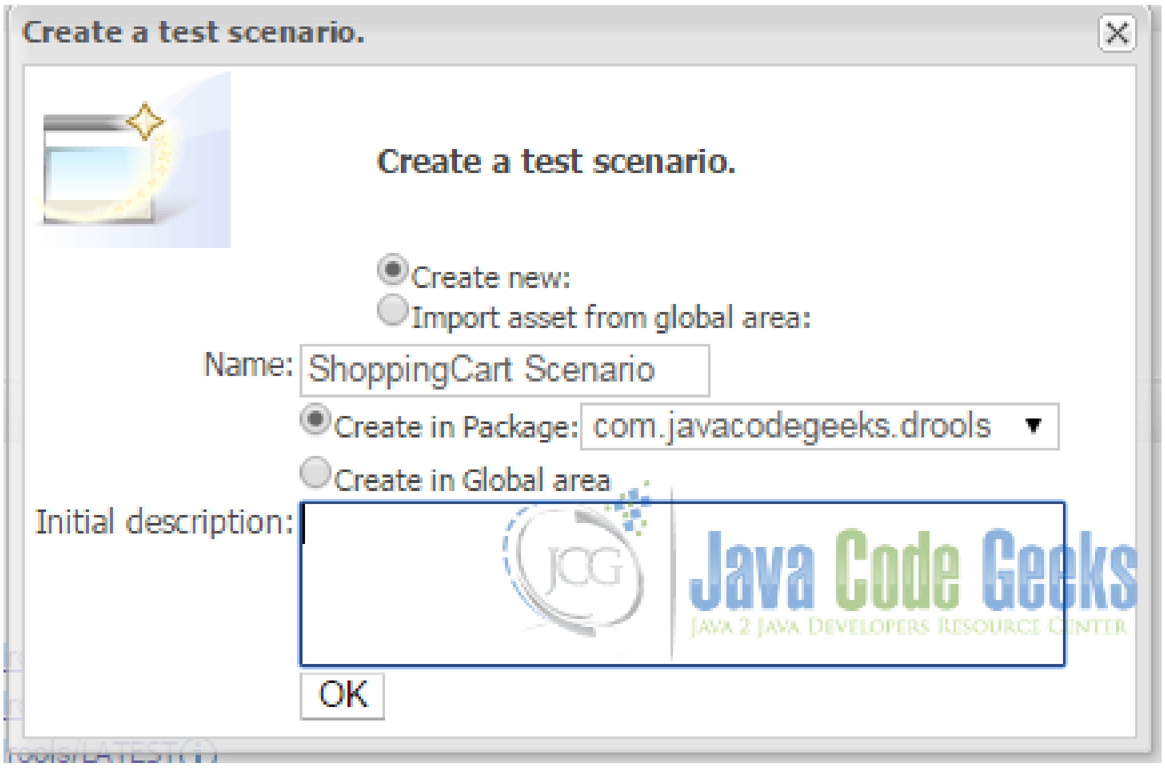
1. Call [cart.addDiscount] double 2

(show options...)

Figure 3.16: Define Rule

3.7 Create and Run the Test Scenario

After defining the rule, we will create a test scenario to test the rule. Select the package and click on *Create new scenario*.


Create a test scenario.

Create a test scenario.

[✓ Create new:] Import asset from global area:

Name: ShoppingCart Scenario

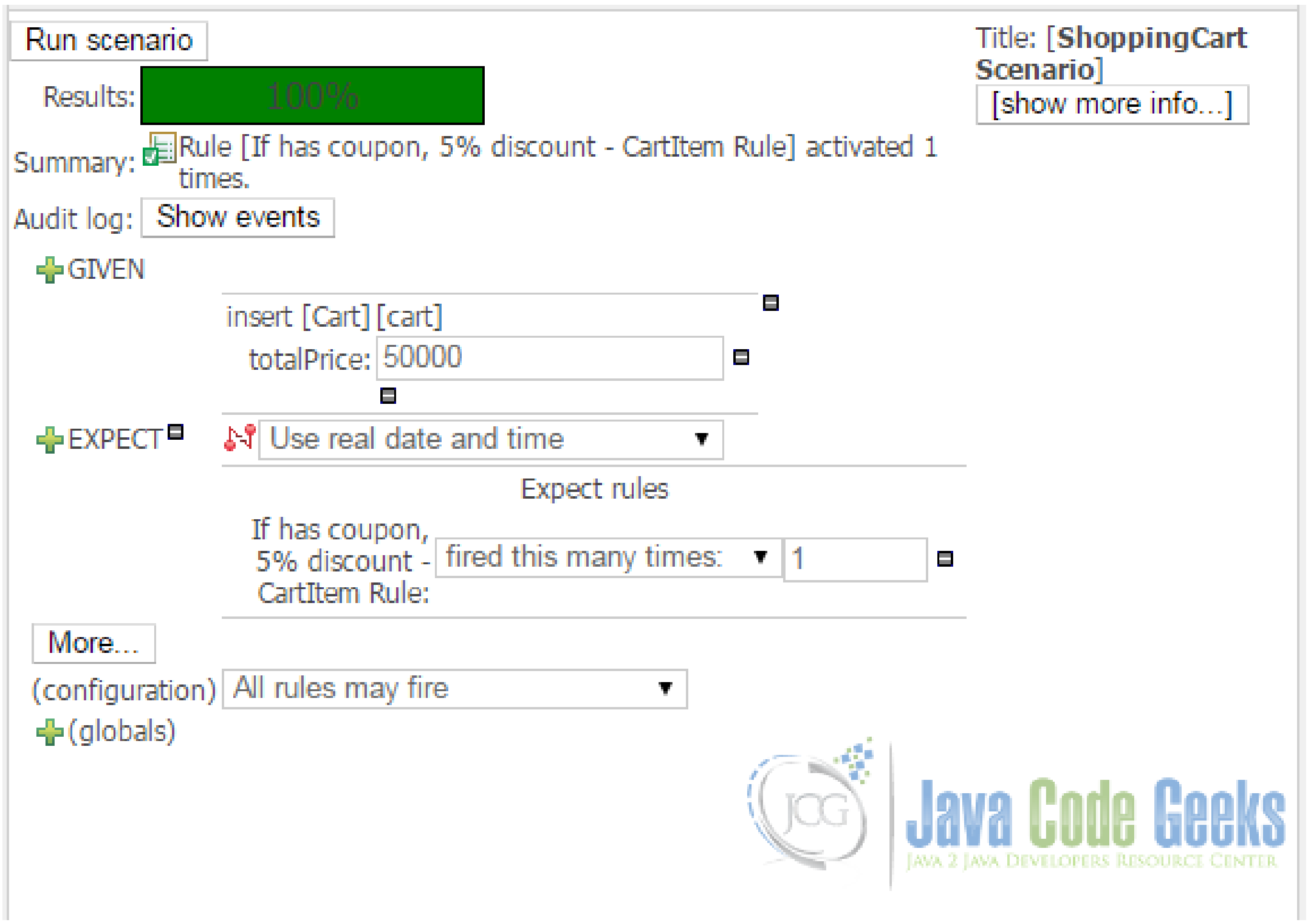
[✓ Create in Package:] com.javacodegeeks.drools

Create in Global area

Initial description:

Figure 3.17: Create Test Scenario

- We will have to click on the + sign to insert a new GIVEN/EXPECT, the small green arrow to refine the scenario, and the - sign to remove.
- We will click on GIVEN to insert a new cart fact and then on Add.
- Next, we will click on the Add a field button that appears. In the dialog box, select the *totalPrice* field.
- Next, we will set a constraint for *totalPrice*, that is, if it is greater than 5000.
- Click on the green + next to EXPECT. In the New Expectation dialog box that is displayed, click on show list and then choose the added rule.
- Change the default (that we expect this rule to fire at least once) to Expect Rules, to fire this many times, and then enter *1* in the new text box that appears.
- Save this test scenario using the button at the top of the screen.
- Finally, we will click on the Run Scenario button, you will get a green bar at the top of the screen saying Results 100%.



Title: [ShoppingCart Scenario]

Results: 100%

Summary: Rule [If has coupon, 5% discount - CartItem Rule] activated 1 times.

Audit log: Show events

GIVEN

insert [Cart] [cart]

totalPrice: 50000

EXPECT

Use real date and time

Expect rules

If has coupon, 5% discount - CartItem Rule: fired this many times: 1

(configuration) All rules may fire

(globals)

Figure 3.18: Run scenario

3.8 Download the Eclipse Project

This was an example about Drools Guvnor.

Download

You can download the full source code of this example here: **DroolsGuvnorExample.zip**

Chapter 4

Drools Expert System Example

In this article, we will see an example of Drools Expert system. First, let's try to understand what is an expert system?

An expert system's goal is to help make a decision or solve a problem. Now to make a proper decision, it relies on are knowledge system and the working memory where we have the data that is to be applied on the knowledge system.

If you want to more know about Drools Introduction or its setup, read here.

This example uses the following frameworks:

- Maven 3.2.3
- Java 8
- Drools 6.2
- Eclipse as the IDE, version Luna 4.4.1.

In your `pom.xml` , you need to add the below dependencies:

- `knowledge-api` - this provides the interfaces and factories
- `drools-core` - this is the core engine, runtime component. This is the only runtime dependency if you are pre-compiling rules.
- `drools-complier` - this contains the compiler/builder components to take rule source, and build executable rule bases. You don't need this during runtime, if your rules are pre-compiled.

4.1 Dependencies

pom.xml:

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/ ←
    XMLSchema-instance"
        xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd ←
            /maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.javacodegeeks.drools</groupId>
        <artifactId>droolsHelloWorld</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <dependencies>
                <dependency>
                        <groupId>org.drools</groupId>
                        <artifactId>drools-compiler</artifactId>
                        <version>${drools.version}</version>
                </dependency>
        </dependencies>
        <properties>
                <drools.version>6.2.0.Final</drools.version>
                <jbpm.version>6.2.0.Final</jbpm.version>
        </properties>
</project>
```

4.2 Expert System

Expert Systems use Knowledge representation to facilitate the codification of knowledge into a knowledge base which can be used for reasoning, i.e., we can process data from the working memory with this knowledge base to infer conclusions. The knowledge system is composed of analytical rules defined by experts. The next diagram basically represents the structure of an expert system. Drools is a Rule Engine that uses the rule-based approach to implement an Expert System. The facts and data are applied against Production Rules to infer conclusions which result in actions. The process of matching the new or existing facts against Production Rules is called Pattern Matching, which is performed by the Inference Engine. The inference engine models in the lines of human reasoning process.

4.3 Rule Structure

A rule is composed of two main structures.

```
when
    <conditions>
then
    <actions>;
```

For example,

```
rule "Add Processed CartItem to Order"
when
    $cartItem : CartItem(cartStatus == CartStatus.PROCESSED)
then
    System.out.println("\\nAdd Processed CartItem to Order Rule");
end
```

4.4 Inference Engine

An inference engine follows the following steps to figure out the rules to apply:

- Inference engine depends on two set of memory, the production memory to access the rules and the working memory to access the facts.
- Facts are asserted into the working Memory where they may then be modified or retracted. We will see an example of this.

```
rule "Print Cart Issues" salience -1
when
$cart : Cart()
then
if ($cart.hasIssues()) {
System.out.println("\\nPrint Cart Issues Rule");
System.out.println("**************************************");
System.out.println($cart.getCartIssues());
insert($cart.getPendingItems());
}
end
```

- A system with a large number of rules and facts may result in many rules being true for the same fact assertion; these rules are said to be in conflict. Inference uses a Conflict Resolution strategy to resolve the order in which rules need to be fired.

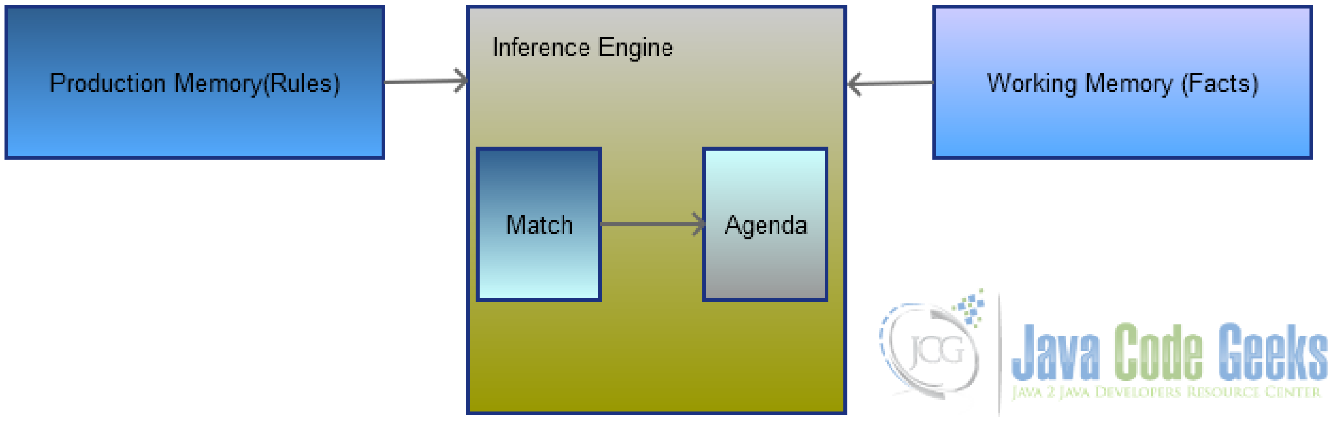


Figure 4.1: Drools Inference Engine

4.5 Forward Chaining

Drools implements and extends the Rete algorithm. It relies on forward chaining. **What is a forward chaining?**

Forward chaining is "data-driven", with facts being asserted into working memory, which results in one or more rules being concurrently true and scheduled for execution by the Agenda. In short, we start with a fact, it propagates and we end in a conclusion.

Here is a forward chain flow:

- Cart Item is Processed.
- If the cart item is processed, create an order for it.
- Since cart item is already processed, result would be *Create an Order for the processed cart item*.

Backward chaining is "goal-driven", meaning that we start with a conclusion which the engine tries to satisfy.

In case of backward chain, the above would look like:

- The order is to be created for a cart item.
- If a cart item is processed, create an order.
- Result in this case would be it picks up cart items that are already processed.

Drools plans to provide support for Backward Chaining in a future release.

Here is an example. Assume product qty is 0, inference engine will end up selecting both *Is Out-Of Stock* rule and *Verify Qty*.

```
package com.javacodegeeks.drools;
import com.javacodegeeks.drools.Cart;
import com.javacodegeeks.drools.CartItem;
import com.javacodegeeks.drools.CartStatus;
import com.javacodegeeks.drools.Product;
import java.util.List;

global List outOfStockProducts;

function String pendingItemKey(CartItem cartItem) {
return cartItem.getCart().getCustomer().getId() + "-"+ cartItem.getProduct().getDesc();
}
//Is it out of stock?
rule "Is Out-Of Stock"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED && product.getAvailableQty() == 0)
then
System.out.println("\\nIs Out-Of Stock Rule");
System.out.println("************************************");
String error = "Can't process as " +  $cartItem.getProduct().getDesc() + " is Out-Of-Stock" ↩
     ;
System.out.println(error);
$cartItem.setErrors(true);
$cartItem.setError(error);
$cartItem.getCart().logItemError(pendingItemKey($cartItem), $cartItem);
outOfStockProducts.add($cartItem.getProduct());
end

rule "Verify Qty"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, qty > product.getAvailableQty())
then
System.out.println("\\nVerify Qty Rule");
System.out.println("************************************");
String error = "Can't process as only " +  $cartItem.getProduct().getAvailableQty() + " of  ↩
    "
+ $cartItem.getProduct().getDesc() + " are left whereas qty requested is " + $cartItem. ↩
    getQty();
System.out.println(error);
$cartItem.setErrors(true);
$cartItem.setError(error);
$cartItem.getCart().logItemError(pendingItemKey($cartItem), $cartItem);
end
```

4.6 Rule Engine

Here are some important points about rule engine.

- Rule engine is all about declarative programming. We only declare what is to be done.
- The how part is based on the data and the behavior which is decoupled from the rules.
- Rule engines allow you to say "What to do", not "How to do it".
- Since each problem is a rule, it improves readability.
- Your data is in your domain objects, the logic is in the rules. This is adavantage if there are many rules and one wants the flexibility of adding rules without changing the existing system.

- You can have rules in more than one file, this way its easy to manage rules in case you have many.
- Finally, we end up creating a repository of knowledge which is executable.
- The rules also serve as documentation as they are have better readability than code.

4.7 Drools Expert System Example

We will use an example of shopping cart which contains cart and cart items. Customer will add one or more products to the cart. There are certain rules that we want to fire as we process the cart.

Rules are:

- If a product needs registration, customer need to register else the item will not be processed.
- There will be discounts applied to the cart total price. If the customer has just registered to the site, there will be a 2% discount on the first purchase.
- If the customer has a coupon, another 5% discount will be applied on the total price. The coupon code and the percentage amounts may vary.
- If customer's requested quantity of product exceeds the available stock, then it will be registered as an issue.
- If a product turns out-of-stock, an error will be registered.

Now since the rules are cart and cart item based, we have grouped the rules in two different files.

Few points about cart rules:

- If a product is not available, the inference engine will end up matching more than one rule *Is Out-Of Stock* and *Verify Qty*.
- If a custom is new and is interested to buy a product which requires mandatory special registration then again we will end up with more than one rule.
- Most of the rules fire for cart item which not yet *PROCESSED* but there is one rule *Add Processed CartItem to Order* which fires ONLY for items which are PROCESSED.

cartItem.drl:

```
package com.javacodegeeks.drools;
import com.javacodegeeks.drools.Cart;
import com.javacodegeeks.drools.CartItem;
import com.javacodegeeks.drools.CartStatus;
import com.javacodegeeks.drools.Product;
import java.util.List;

global List<Product> outOfStockProducts;

function String pendingItemKey(CartItem cartItem) {
return cartItem.getCart().getCustomer().getId() + "-"+ cartItem.getProduct().getDesc();
}
//Is it out of stock?
rule "Is Out-Of Stock"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED && product.getAvailableQty() == 0)
then
System.out.println("\\nIs Out-Of Stock Rule");
System.out.println("************************************");
String error = "Can't process as " +  $cartItem.getProduct().getDesc() + " is Out-Of-Stock" ↩
     ;
System.out.println(error);
```

```
$cartItem.setErrors(true);
$cartItem.setError(error);
$cartItem.getCart().logItemError(pendingItemKey($cartItem), $cartItem);
outOfStockProducts.add($cartItem.getProduct());
end

rule "Verify Qty"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, qty > product.getAvailableQty())
then
System.out.println("\\nVerify Qty Rule");
System.out.println("************************************");
String error = "Can't process as only " +  $cartItem.getProduct().getAvailableQty() + " of  ↩
    "
+ $cartItem.getProduct().getDesc() + " are left whereas qty requested is " + $cartItem. ↩
    getQty();
System.out.println(error);
 $cartItem.setErrors(true);
$cartItem.setError(error);
$cartItem.getCart().logItemError(pendingItemKey($cartItem), $cartItem);
end

rule "If has coupon, 5% discount"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, cart.customer.coupon == 'DISC01')
then
if (!$cartItem.hasErrors()) {
    System.out.println("\\nIf has coupon, 5% discount Rule");
    System.out.println("************************************");
    double discount = ((double)$cartItem.getCart().getTotalPrice())*0.05d;
    System.out.println("Coupon Rule: Process " + $cartItem.getProduct() + ", qty " +  ↩
        $cartItem.getQty() + ", apply discount " + discount);
    $cartItem.getCart().addDiscount(discount);
}
end

rule "If new, 2% discount"
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, cart.customer.isNew())
then
if (!$cartItem.hasErrors()) {
    System.out.println("\\nIf new, 2% discount Rule");
    System.out.println("************************************");
    double discount = ((double)$cartItem.getCart().getTotalPrice())*0.2d;
    System.out.println("New Customer Rule: Process " + $cartItem.getProduct() + ", qty " +  ↩
        $cartItem.getQty() + ", apply discount " + discount);
    $cartItem.getCart().addDiscount(discount);
}
end

rule "Has customer registered for the product?" salience 1
when
$cartItem : CartItem(cartStatus != CartStatus.PROCESSED, product.isRequiresRegisteration(), ↩
     !cart.customer.isRegistered(product))
then
System.out.println("\\nHas customer registered for the product? Rule");
System.out.println("************************************");
String error = "Can't process " + $cartItem.getProduct() + ", as requires registration.  ↩
    Customer not registered for the product!";
System.out.println(error);
$cartItem.setErrors(true);
$cartItem.setError(error);
```

```
$cartItem.getCart().logItemError(pendingItemKey($cartItem), $cartItem);
end

rule "Add Processed CartItem to Order"
when
$cartItem : CartItem(cartStatus == CartStatus.PROCESSED)
then
System.out.println("\\nAdd Processed CartItem to Order Rule");
System.out.println("************************************");
System.out.println("Add to order " + $cartItem);
end
```

Let's review cart rules.

- Cart Rules, insert and modify the facts.
- Once all items are processed, cart goes through each one to figure out if they have an issue. If yes, it inserts a new fact *PendingItems* into the working memory. See *Print Cart Issues*
- If the items have no issues, those items get marked as *PROCESSED* and the CartItem fact gets updated. This results into the rules getting re-evaluated. See *Mark the items processed*
- There is one rule in cartItem rules file which work on the PROCESSED cart item. Once the CartItem fact is updated, this rule comes into picture and creates an order item for the cart item. See *Add Processed CartItem to Order*

Here are the cart rules.

cart.drl:

```
package com.javacodegeeks.drools;
import com.javacodegeeks.drools.Cart;
import com.javacodegeeks.drools.CartItem;
import com.javacodegeeks.drools.CartStatus;

rule "Print Cart Issues" salience -1
when
$cart : Cart()
then
if ($cart.hasIssues()) {
System.out.println("\\nPrint Cart Issues Rule");
System.out.println("************************************");
System.out.println($cart.getCartIssues());
insert($cart.getPendingItems());
}
end

rule "Print Pending Items"
when
$pendingItems : PendingItems()
then
System.out.println("\\nPrint Pending Items Rule");
System.out.println("************************************");
for (CartItem cartItem : $pendingItems.getCartItems()) {
System.out.println(cartItem);
}
end

rule "Mark the items processed" salience -2
when
$cart : Cart()
then
System.out.println("\\nMark the items processed Rule");
System.out.println("************************************");
for (CartItem cartItem : $cart.getCartItems()) {
if (cartItem.getCartStatus() != CartStatus.NEW || cartItem.getCartStatus() != CartStatus. ↩
    PENDING) {
cartItem.updateAsProcessed();
System.out.println(cartItem + " is processed");
update(cartItem);
}
}
end
```

Let’s run the example.

```
package com.javacodegeeks.drools;

import java.util.ArrayList;
import java.util.List;

import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

/**
 * This is a sample class to launch a rule.
 */
public class DroolsExpertSystemExample {

    public static final void main(String[] args) {
        try {
            // load up the knowledge base
            KieServices ks = KieServices.Factory.get();
            KieContainer kContainer = ks.getKieClasspathContainer();
            KieSession kSession = kContainer.newKieSession("ksession-rules");

            Customer customer = Customer.newCustomer("RS");
                Product p1 = new Product("Laptop", 15000);
                Product p2 = new Product("Mobile", 5000);
                p2.setRequiresRegistration(true);
                Product p3 = new Product("Books", 2000);

                Product p4OutOfStock = new Product("TV", 2000);
                p4OutOfStock.setAvailableQty(0);

                Product p5 = new Product("Tab", 10000);
                p5.setAvailableQty(2);

                customer.addItem(p1, 1);
                customer.addItem(p2, 2);
                customer.addItem(p3, 5);
                customer.setCoupon("DISC01");

                List<CartItem> cartItems = customer.getCart().getCartItems();
                for (CartItem cartItem: cartItems) {
                        kSession.insert(cartItem);
                }
                System.out.println("************* Fire Rules **************");
            kSession.fireAllRules();
            System.out.println("************************************");
            System.out.println("Customer cart\\n" + customer);

            Customer newCustomer = Customer.newCustomer("JOHN01");
                newCustomer.addItem(p1, 1);
                newCustomer.addItem(p2, 2);
                newCustomer.addItem(p4OutOfStock, 1);
                newCustomer.addItem(p5, 10);

                cartItems = newCustomer.getCart().getCartItems();
                for (CartItem cartItem: cartItems) {
                        kSession.insert(cartItem);
                }
                kSession.insert(newCustomer.getCart());
                kSession.setGlobal("outOfStockProducts", new ArrayList<Product>());
                System.out.println("************* Fire Rules **************");
            kSession.fireAllRules();
            System.out.println("************************************");
            System.out.println("Customer cart\\n" + customer);

        } catch (Throwable t) {
            t.printStackTrace();
        }
    }
}
```

Output:

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
************* Fire Rules **************

Has customer registered for the product? Rule
************************************
Can't process product: Mobile, price: 5000, as requires registration. Customer not  ↩
    registered for the product!

If has coupon, 5% discount Rule
************************************
Coupon Rule: Process product: Books, price: 2000, qty 5, apply discount 1250.0

If has coupon, 5% discount Rule
************************************
Coupon Rule: Process product: Laptop, price: 15000, qty 1, apply discount 1250.0

If new, 2% discount Rule
************************************
New Customer Rule: Process product: Books, price: 2000, qty 5, apply discount 5000.0

If new, 2% discount Rule
************************************
New Customer Rule: Process product: Laptop, price: 15000, qty 1, apply discount 5000.0
************************************
Customer cart
Customer new? true
Coupon: DISC01
Product: product: Laptop, price: 15000, qty: 1, processed: false
Product: product: Mobile, price: 5000, qty: 2, processed: true, Issue: Can't process  ↩
    product: Mobile, price: 5000, as requires registration. Customer not registered for the  ↩
    product!
Product: product: Books, price: 2000, qty: 5, processed: false
Discount: 12500.0
Total: 25000
Total After Discount: 12500
************* Fire Rules **************
```

```
Has customer registered for the product? Rule
************************************
Can't process product: Mobile, price: 5000, as requires registration. Customer not  ↩
    registered for the product!

Is Out-Of Stock Rule
************************************
Can't process as TV is Out-Of-Stock

Verify Qty Rule
************************************
Can't process as only 2 of Tab are left whereas qty requested is 10

Verify Qty Rule
************************************
Can't process as only 0 of TV are left whereas qty requested is 1

If new, 2% discount Rule
************************************
New Customer Rule: Process product: Laptop, price: 15000, qty 1, apply discount 3000.0

Print Cart Issues Rule
************************************
JOHN01-TabProduct: product: Tab, price: 10000, qty: 10, processed: true, Issue: Can't  ↩
    process as only 2 of Tab are left whereas qty requested is 10
JOHN01-MobileProduct: product: Mobile, price: 5000, qty: 2, processed: true, Issue: Can't  ↩
    process product: Mobile, price: 5000, as requires registration. Customer not registered ↩
    for the product!
JOHN01-TVProduct: product: TV, price: 2000, qty: 1, processed: true, Issue: Can't process  ↩
    as only 0 of TV are left whereas qty requested is 1


Print Pending Items Rule
************************************
Product: product: Mobile, price: 5000, qty: 2, processed: true, Issue: Can't process  ↩
    product: Mobile, price: 5000, as requires registration. Customer not registered for the  ↩
    product!
Product: product: TV, price: 2000, qty: 1, processed: true, Issue: Can't process as only 0  ↩
    of TV are left whereas qty requested is 1
Product: product: Tab, price: 10000, qty: 10, processed: true, Issue: Can't process as only ↩
     2 of Tab are left whereas qty requested is 10
Product: product: TV, price: 2000, qty: 1, processed: true, Issue: Can't process as only 0  ↩
    of TV are left whereas qty requested is 1

Mark the items processed Rule
************************************
Product: product: Laptop, price: 15000, qty: 1, processed: false is processed
Product: product: Mobile, price: 5000, qty: 2, processed: true, Issue: Can't process  ↩
    product: Mobile, price: 5000, as requires registration. Customer not registered for the  ↩
    product! is processed
Product: product: TV, price: 2000, qty: 1, processed: true, Issue: Can't process as only 0  ↩
    of TV are left whereas qty requested is 1 is processed
Product: product: Tab, price: 10000, qty: 10, processed: true, Issue: Can't process as only ↩
     2 of Tab are left whereas qty requested is 10 is processed

Add Processed CartItem to Order Rule
************************************
Add to order Product: product: Tab, price: 10000, qty: 10, processed: true, Issue: Can't  ↩
    process as only 2 of Tab are left whereas qty requested is 10

Add Processed CartItem to Order Rule
************************************
```

```
Add to order Product: product: TV, price: 2000, qty: 1, processed: true, Issue: Can't  ↩
    process as only 0 of TV are left whereas qty requested is 1

Add Processed CartItem to Order Rule
************************************
Add to order Product: product: Mobile, price: 5000, qty: 2, processed: true, Issue: Can't  ↩
    process product: Mobile, price: 5000, as requires registration. Customer not registered  ↩
    for the product!

Add Processed CartItem to Order Rule
************************************
Add to order Product: product: Laptop, price: 15000, qty: 1, processed: false
************************************
Customer cart
Customer new? true
Coupon: DISC01
Product: product: Laptop, price: 15000, qty: 1, processed: false
Product: product: Mobile, price: 5000, qty: 2, processed: true, Issue: Can't process  ↩
    product: Mobile, price: 5000, as requires registration. Customer not registered for the  ↩
    product!
Product: product: Books, price: 2000, qty: 5, processed: false
Discount: 12500.0
Total: 25000
Total After Discount: 12500
```

4.8 Download the Eclipse Project

This was an example about Drools Expert System.

Download

You can download the full source code of this example here: **DroolsExpertSystemExample.zip**

Chapter 5

JBoss BRMS Drools Example

In this article, we will see an example of JBoss BRMS. BRMS is a JBoss Rules based server-side solution for the management, storage, editing and deployment of rules. JBoss BRMS 6 is backed by a GIT based repository, which is very much in line with how the mainstream enterprises are storing and working to deploy their projects.

If you want to more know about Drools Introduction or its setup, read here.

This example uses the following frameworks:

- Maven 3.2.3
- JBoss BRMS 6.1.0.GA

5.1 What is a Business Rules Management System (BRMS)?

BRMS is a JBoss Rules based server-side solution for the management, storage, editing and deployment of rules and other JBoss Rules assets. BRMS helps us in managing rules, listed below:

- Manage rules in a multi user environment
- Centralized repository of rules
- Allowing change in a controlled fashion, with user friendly interfaces.
- Manage versions/deployment of rules
- Very useful if have lots of "business" rules

5.2 Users of BRMS

Its audience is a bit wide. It includes Business Analyst, Rule expert, Developer and Administrators. BRMS allows different roles to be assigned to different users based on what the user is going make out of it.

5.3 BRMS Installation

Download latest BRMS from the below link: https://www.jboss.org/products/brms/download/. The version I am using is JBoss BRMS 6.1.0.GA.

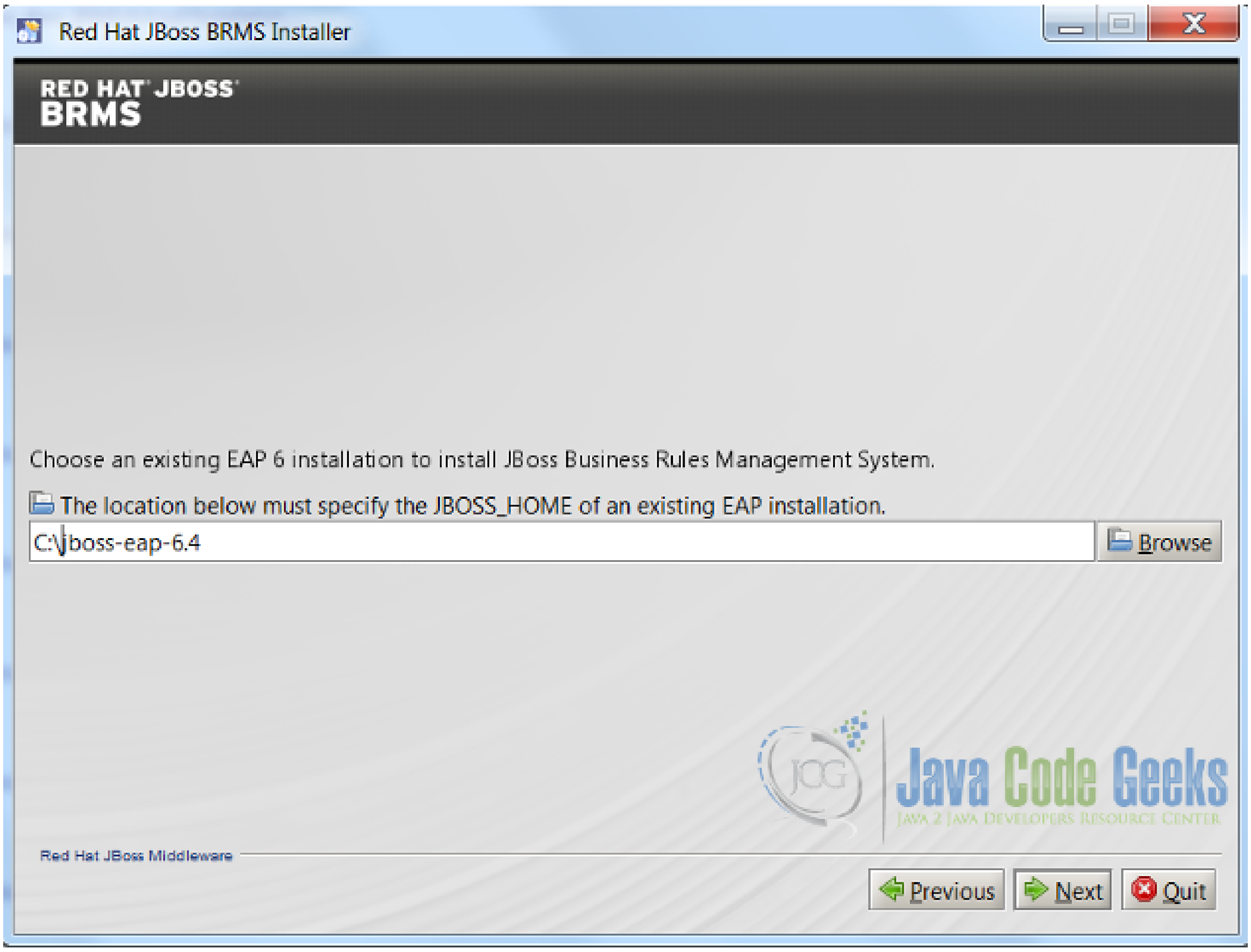
Before you start download make sure you are done with the below two:

- You must have an account with JBoss and logged in before you start the download.
- You must have already installed JBoss EAP else download JBoss EAP 6.4.0.GA Installer and install JBoss EAP.

Download the BRMS 6.1.0 Installer (jboss-brms-6.1.0.GA-installer.jar) to a directory of your choice.

Double click on the installer to run it.

You will be prompted to enter the JBoss EAP Home directory.



Choose an existing EAP 6 installation to install JBoss Business Rules Management System.

The location below must specify the JBOSS_HOME of an existing EAP installation.

C:\jboss-eap-6.4

Figure 5.1: BRMS Installer - Enter JBoss EAP Home

Follow the instructions as you navigate through each page.

You can create an application user using add-user.bat

```
C:\\jboss-eap-6.4\\bin>add-user.bat -a -u 'analyst' -p 'analyst1234!! -ro 'admin,analyst'
```

5.4 Start the JBoss BRMS

First start JBoss EAP Server. Open file explorer, navigate to the root of the JBoss EAP home/bin directory. Double click on standalone.bat.

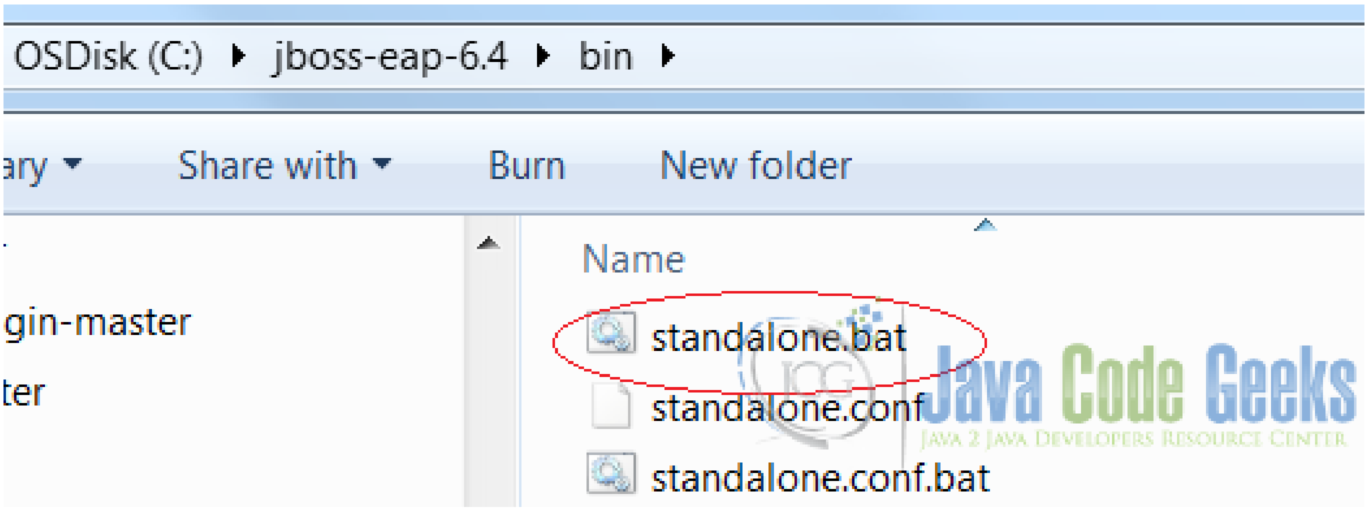


Figure 5.2: Standalone.bat

Navigate to https://localhost:8080/business-central in a web browser.

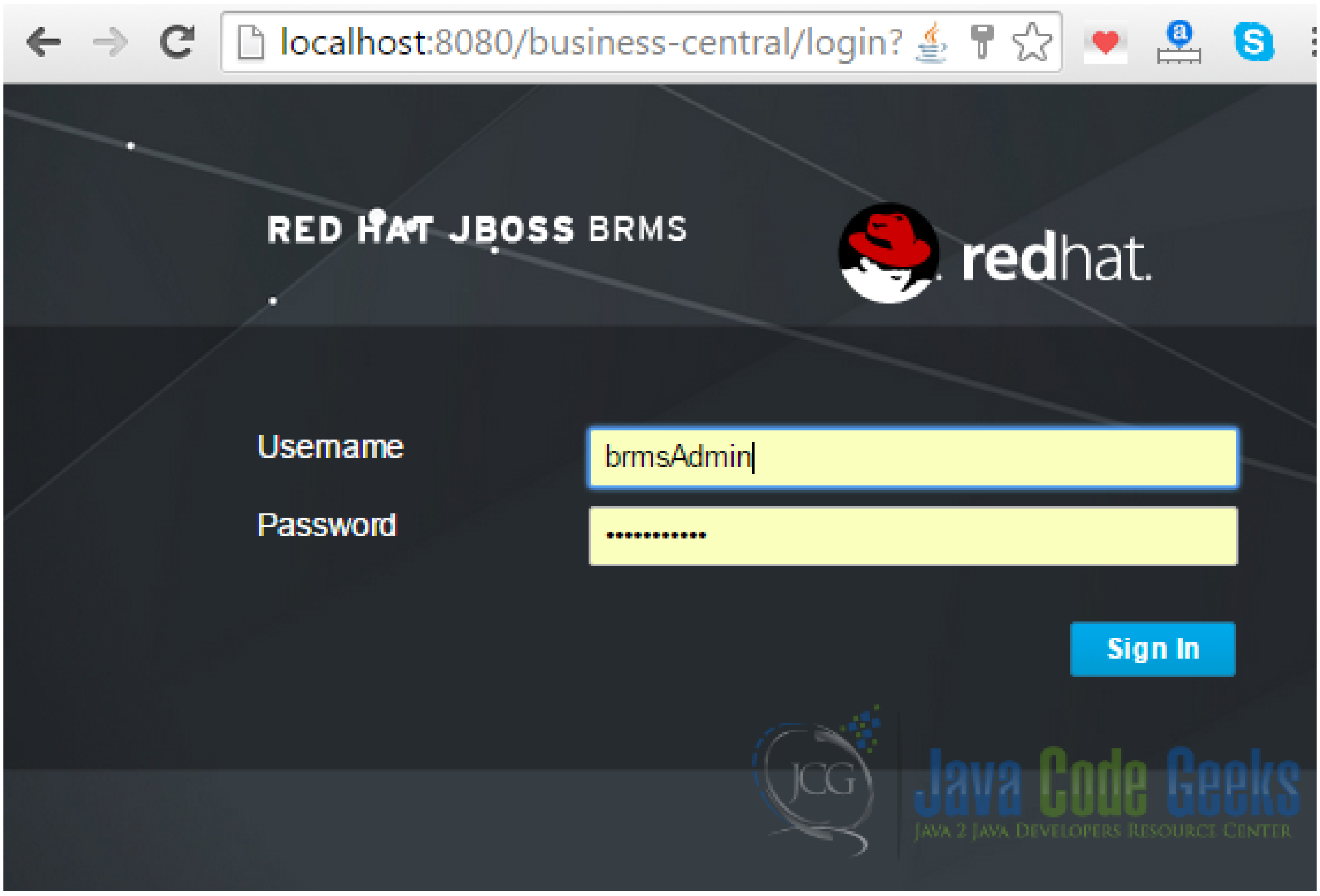
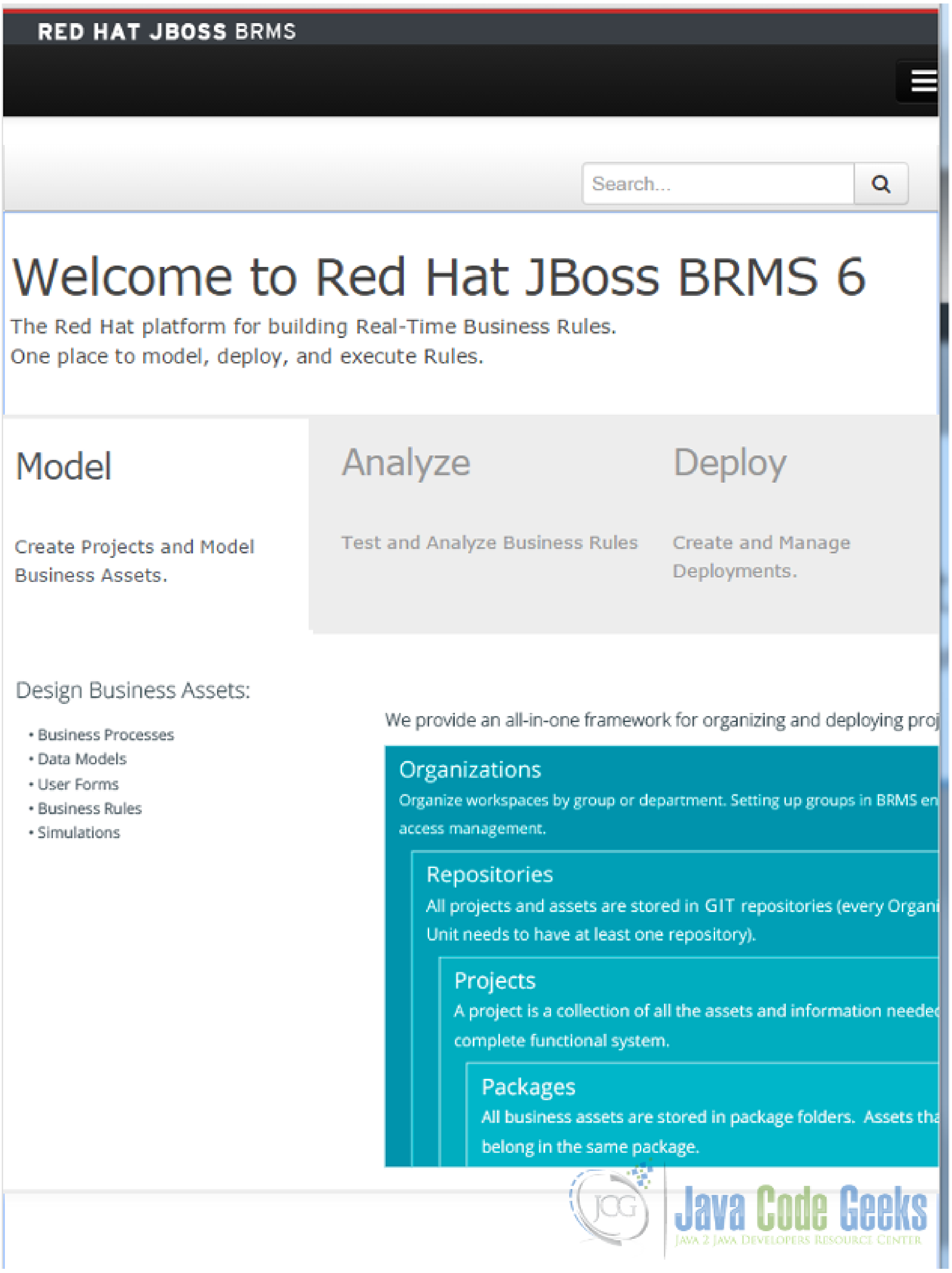


Figure 5.3: JBoss BRMS Login

Once you login, you will be taken to the below home page.



RED HAT JBOSS BRMS

Search...

Welcome to Red Hat JBoss BRMS 6

The Red Hat platform for building Real-Time Business Rules.
One place to model, deploy, and execute Rules.

Model

Create Projects and Model Business Assets.

Analyze

Test and Analyze Business Rules

Deploy

Create and Manage Deployments.

Design Business Assets:

- Business Processes
- Data Models
- User Forms
- Business Rules
- Simulations

We provide an all-in-one framework for organizing and deploying proj

Organizations
Organize workspaces by group or department. Setting up groups in BRMS en
access management.

Repositories
All projects and assets are stored in GIT repositories (every Organi
Unit needs to have at least one repository).

Projects
A project is a collection of all the assets and information needed
complete functional system.

Packages
All business assets are stored in package folders. Assets tha
belong in the same package.

Figure 5.4: JBoss BRMS Home

5.5 Import BRMS Repository

From main menu, Click on:

- Authoring → Administration:

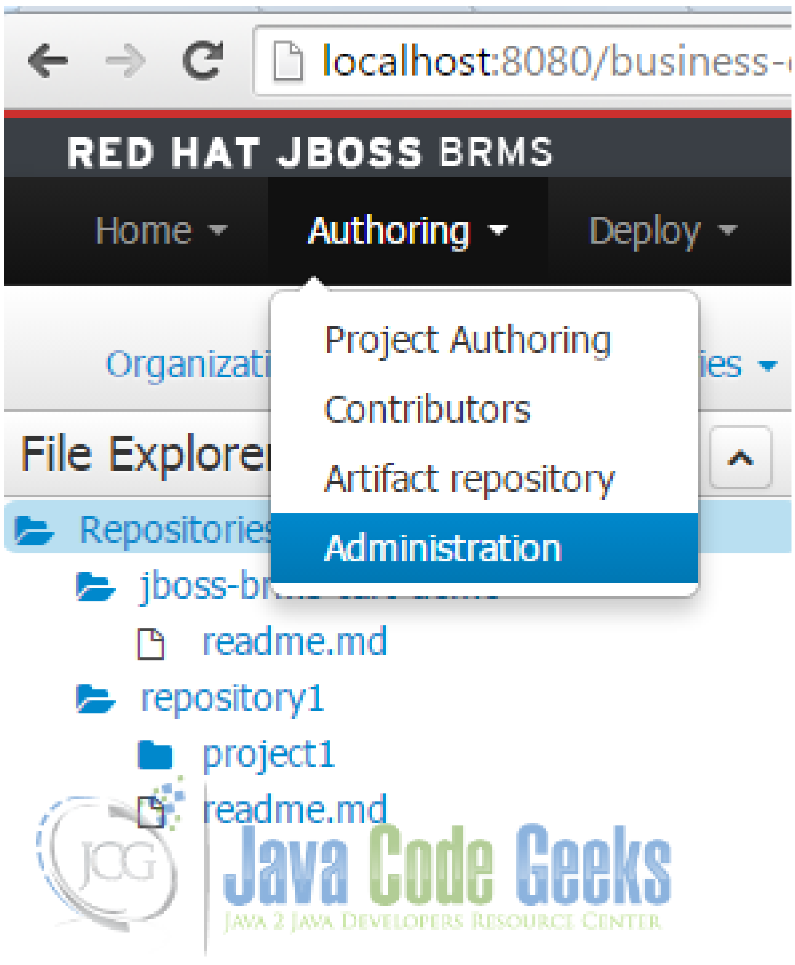


Figure 5.5: BRMS Authoring→Administration

- Next, click on Repositories → Clone Repository:

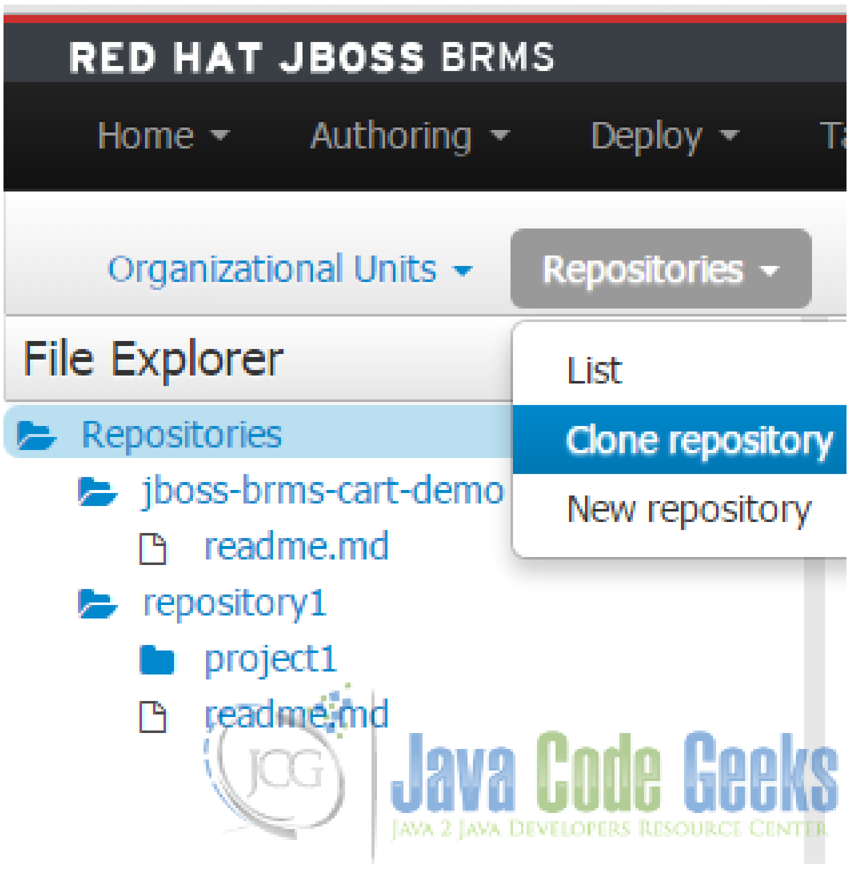
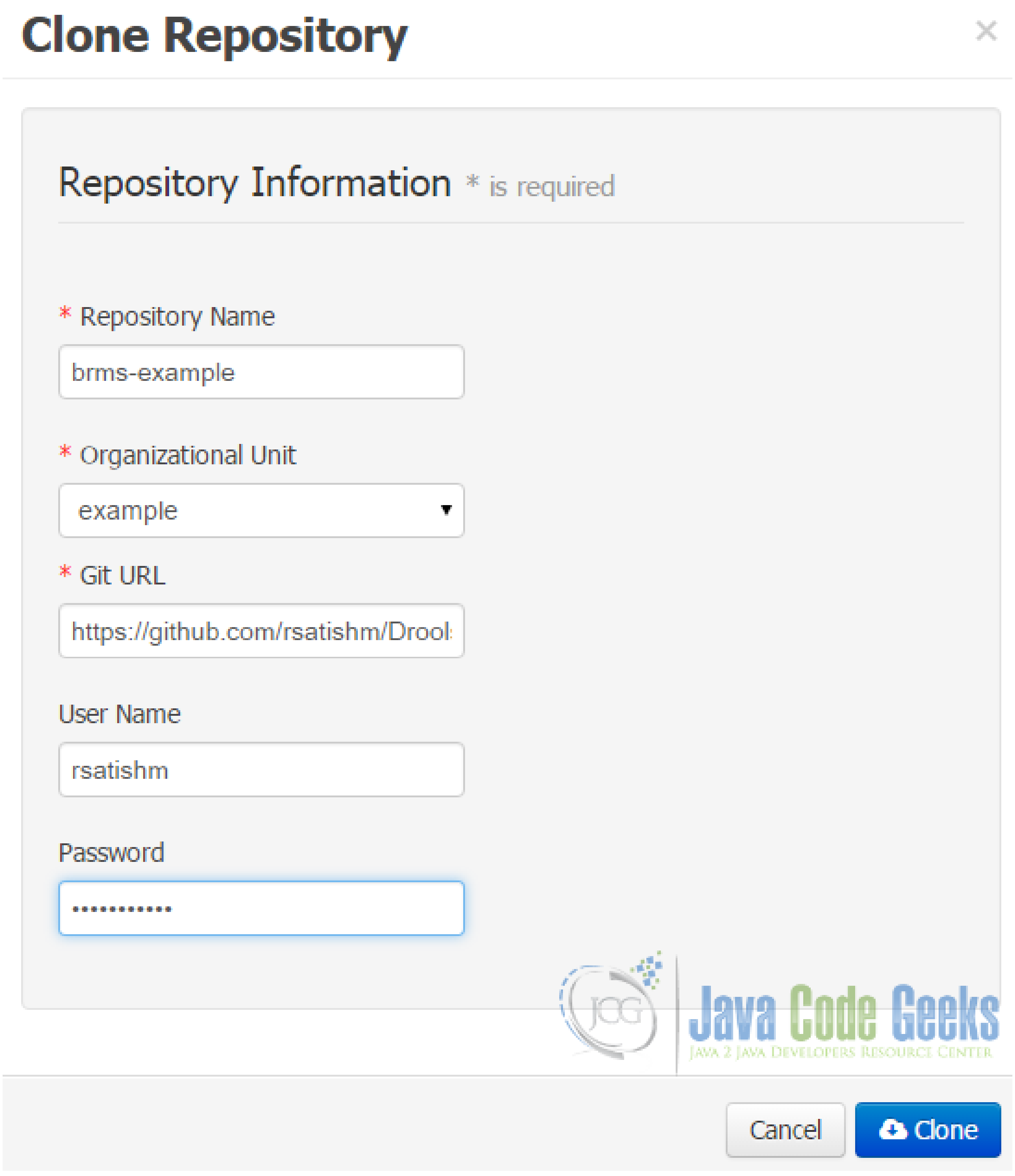


Figure 5.6: BRMS Repositories→Clone Repository

- In pop-up window, provide following:

Repository name: brms-example **Organizational unit:** Example **Git URL:** https://github.com/rsatishm/DroolsBrmsExample.git



Clone Repository

Repository Information * is required

* Repository Name

brms-example

* Organizational Unit

example

* Git URL

https://github.com/rsatishm/Drool

User Name

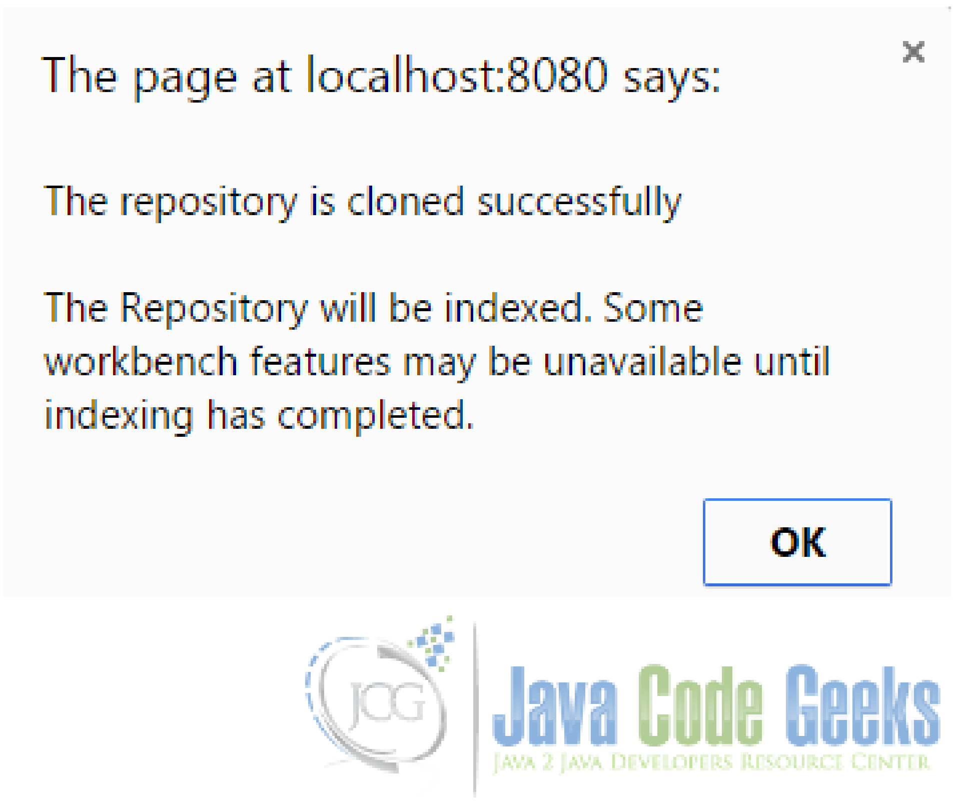
rsatishm

Password

••••••••••••

Figure 5.7: BRMS Clone Repository

- After clone you should see below message:

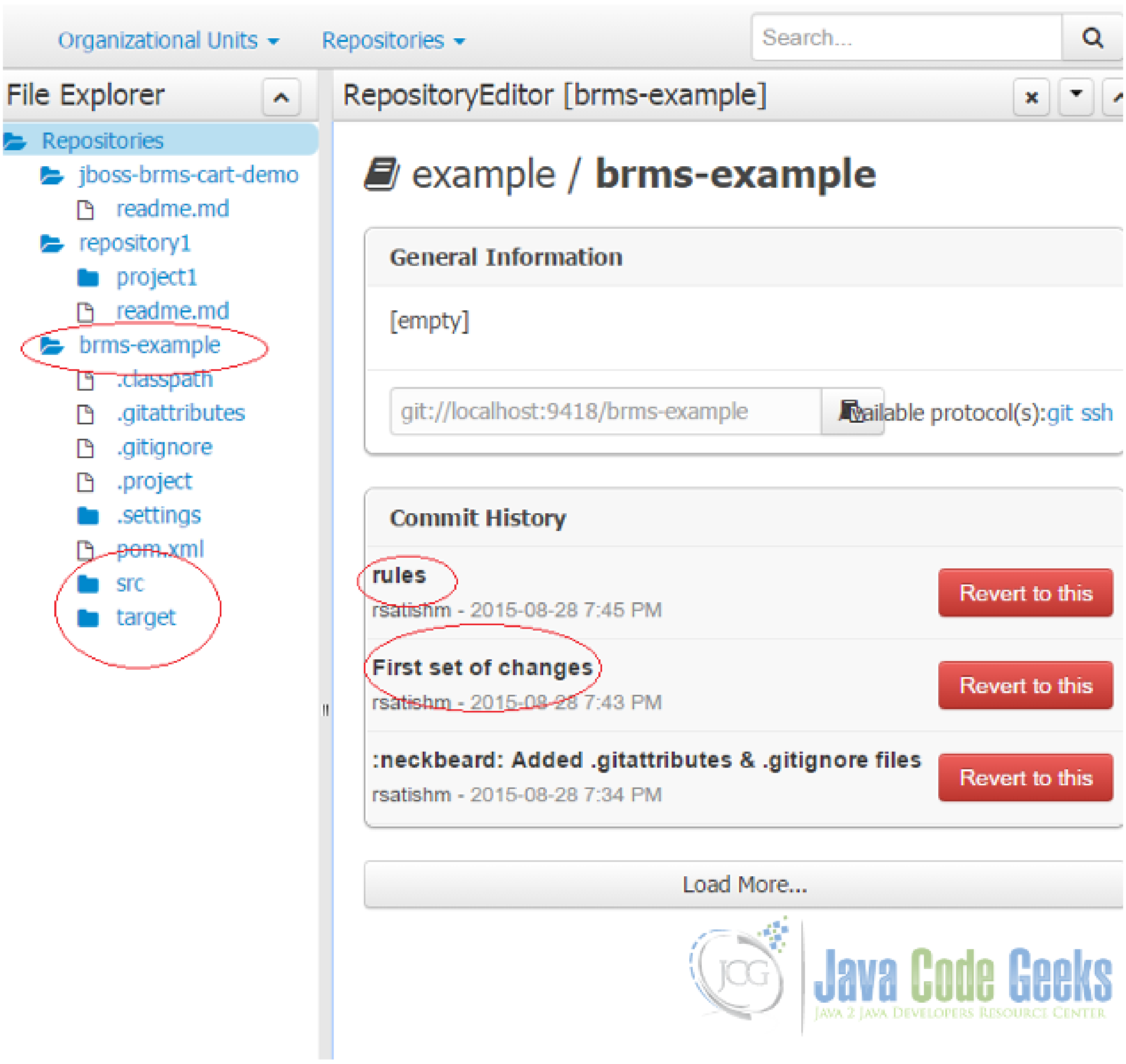

The page at localhost:8080 says:

The repository is cloned successfully

The Repository will be indexed. Some workbench features may be unavailable until indexing has completed.

Figure 5.8: BRMS After Clone Message

- You can see the repository structure on the left side and details in the right panel.



rules
rsatishm - 2015-08-28 7:45 PM

First set of changes
rsatishm - 2015-08-28 7:43 PM

:neckbeard: Added .gitattributes & .gitignore files
rsatishm - 2015-08-28 7:34 PM

Figure 5.9: BRMS Project Repository

5.6 Create Project

Navigate to Authoring→Project Authoring. Click on New Item→Project.

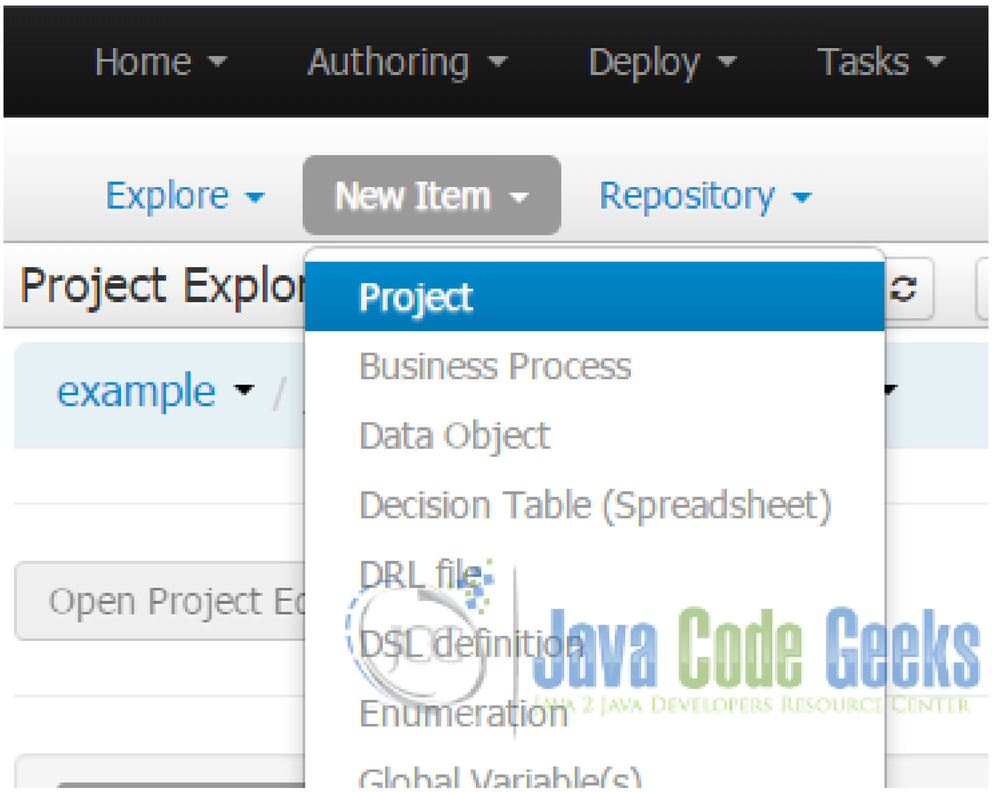


Figure 5.10: BRMS - Create Project

Enter a project name and click on Ok.

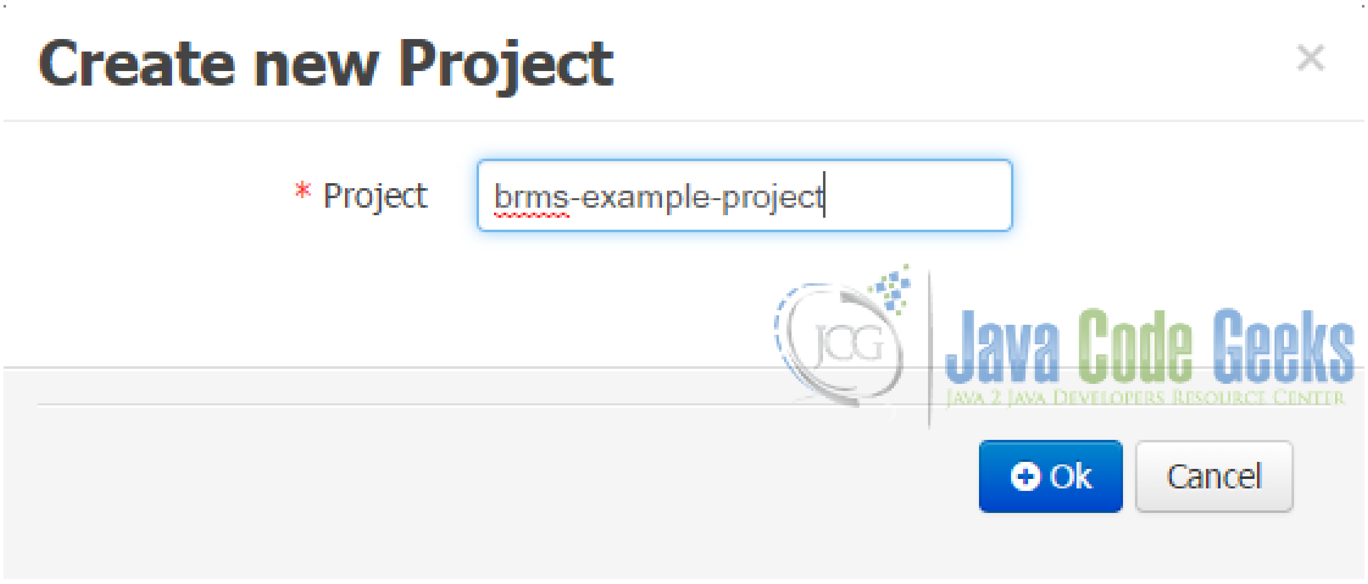
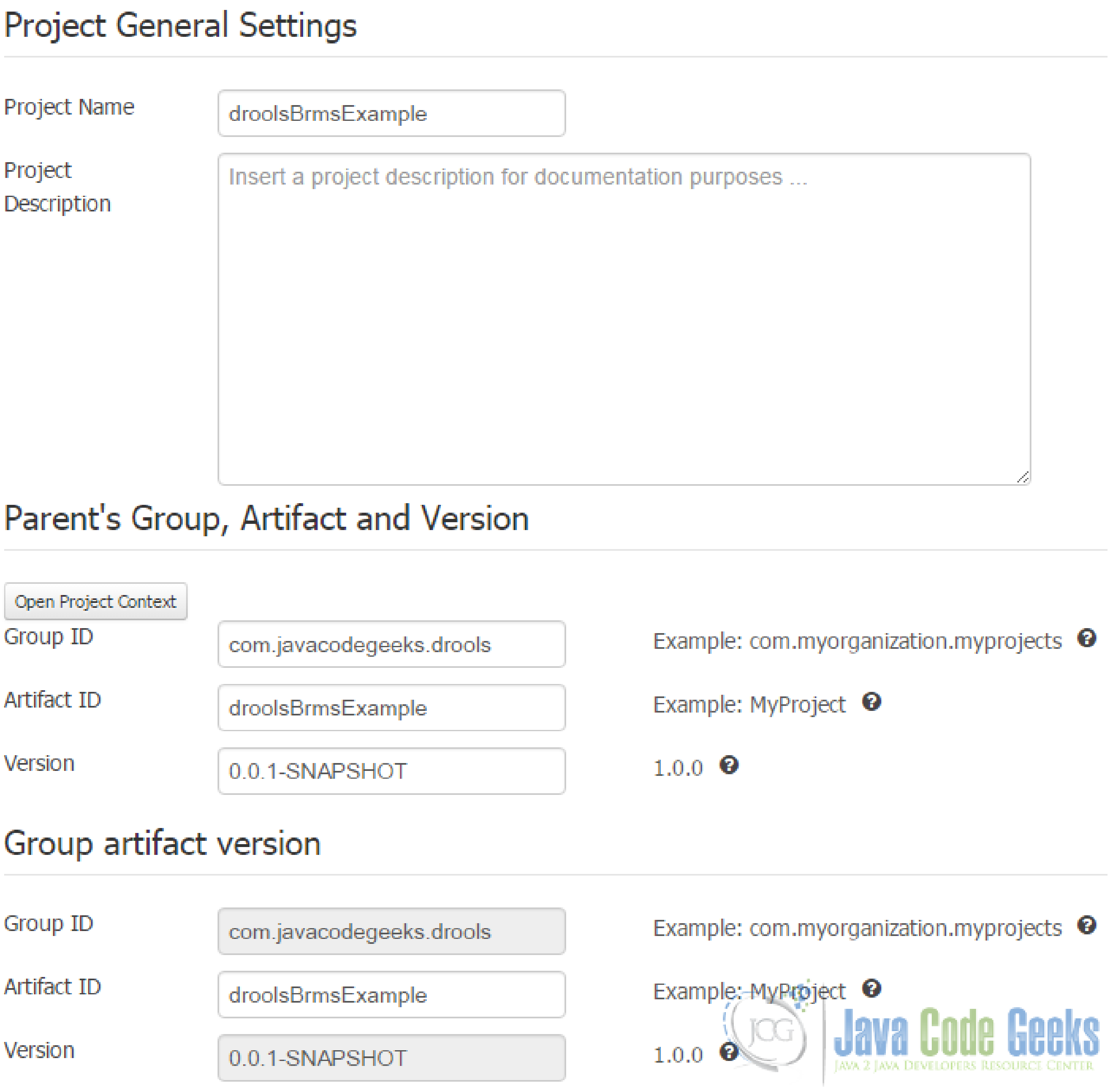


Figure 5.11: BRMS - Create new Project

Once saved, you can see the project details.



Project General Settings

Project Name	droolsBrmsExample
Project Description	Insert a project description for documentation purposes ...

Parent's Group, Artifact and Version

Group ID	com.javacodegeeks.drools	Example: com.myorganization.myprojects
Artifact ID	droolsBrmsExample	Example: MyProject
Version	0.0.1-SNAPSHOT	1.0.0

Group artifact version

Group ID	com.javacodegeeks.drools	Example: com.myorganization.myprojects
Artifact ID	droolsBrmsExample	Example: MyProject
Version	0.0.1-SNAPSHOT	1.0.0

Figure 5.12: BRMS - Project Details

5.7 Download the source code

This was an example about JBoss Drools BRMS.

Download

You can download the full source code of this example here: **DroolsBrmsExample.zip**

Chapter 6

Drools Backward Chaining Example

In this article, we will see a little introduction and example of what is backward chaining and how to implemente it with jboss drools.

This example uses the following technologies and frameworks:

- Maven 3.3.9
- Jboss Studio 10.3
- Drools Engine 7.0
- JDK 1.8.0_71

6.1 Introduction

Backward chaining, is a concept that allows in a graph structure(derivation tree) to get through each node or find the possible paths between 2 nodes, using recursion and reacting to changing values. The latter is because drools is a reactive rule engine. Drools support 2 types of operations push (reactive operation) and pull (data query). So basically the backward chaining is how each node is connected to his parent and take advantage of that to find relations between them. We will achieve this on this example, by using drools and recursion queries that allows to search on derivation tree structure.

6.1.1 Example Graph

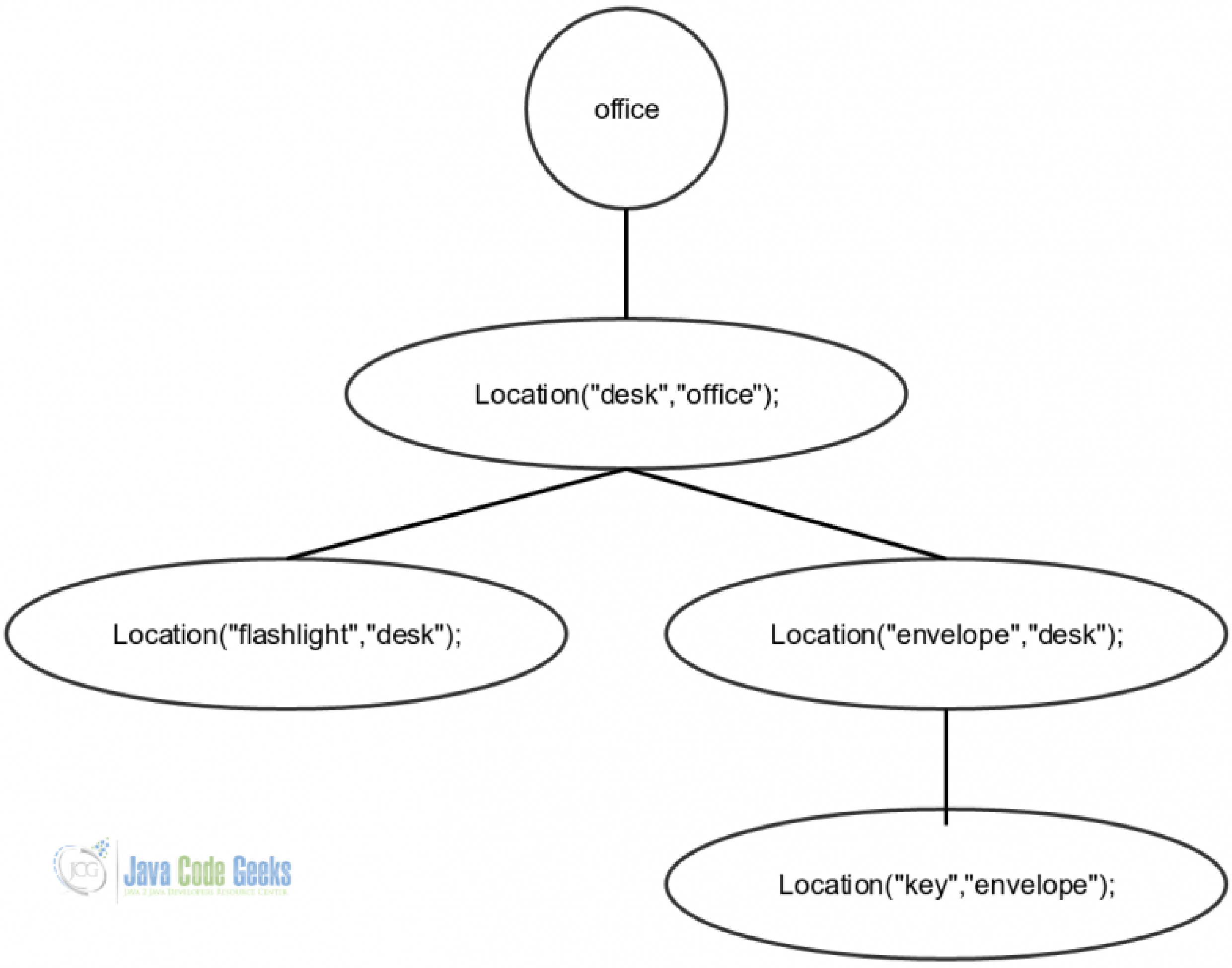

office
Location("desk","office");
Location("flashlight","desk");
Location("envelope","desk");
Location("key","envelope");
Java Code Geeks

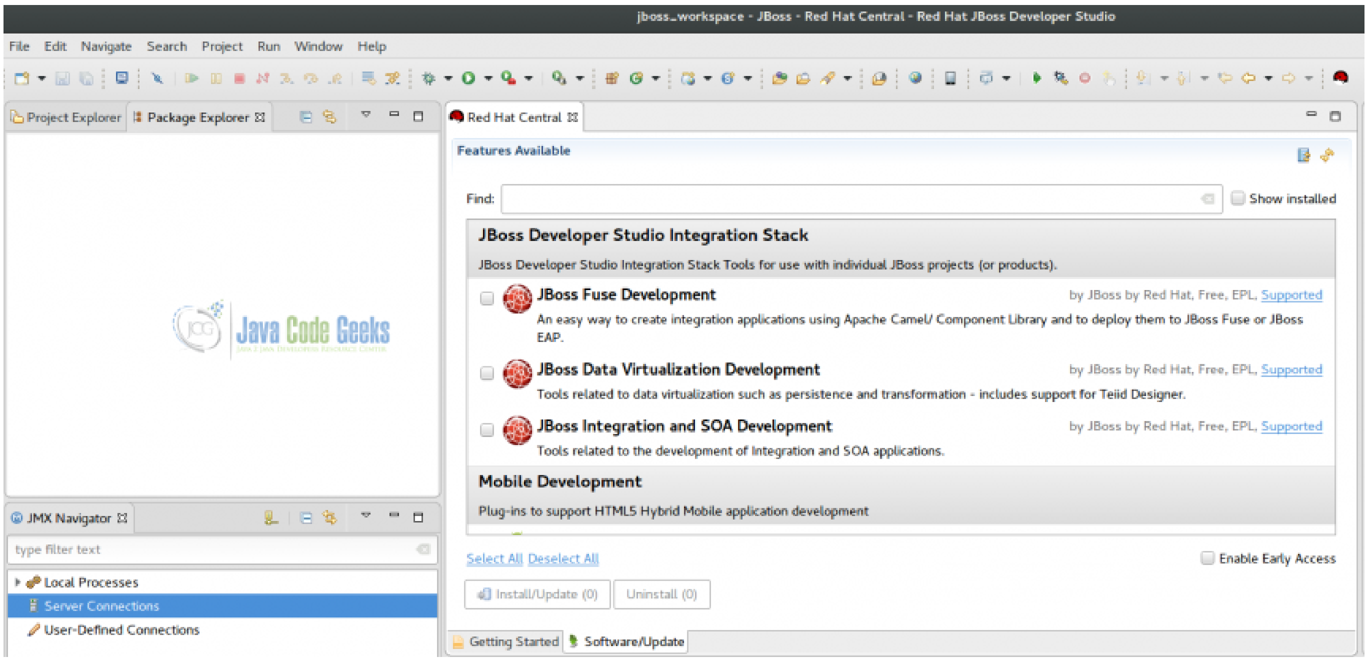
Figure 6.1: Backward chaining graph

This graph is to illustrate the data that we will be using on this example to apply backward chaining and determinate that the key is in the envelope.

6.2 Configure Necessary Tools

Before we continue, it is necessary to install the jboss developer studio in order to build the drools project that implement the backward chaining using recursion, base on the graph on Figure 5.1. To download these tools go to jboss devstudio site and install them.

Next Add the drools plugin to the IDE, open the jboss IDE and select the bottom tab called software update as it's shown below.



JBoss Developer Studio Integration Stack
JBoss Developer Studio Integration Stack Tools for use with individual JBoss projects (or products).

JBoss Fuse Development — by JBoss by Red Hat, Free, EPL, Supported
An easy way to create integration applications using Apache Camel/ Component Library and to deploy them to JBoss Fuse or JBoss EAP.

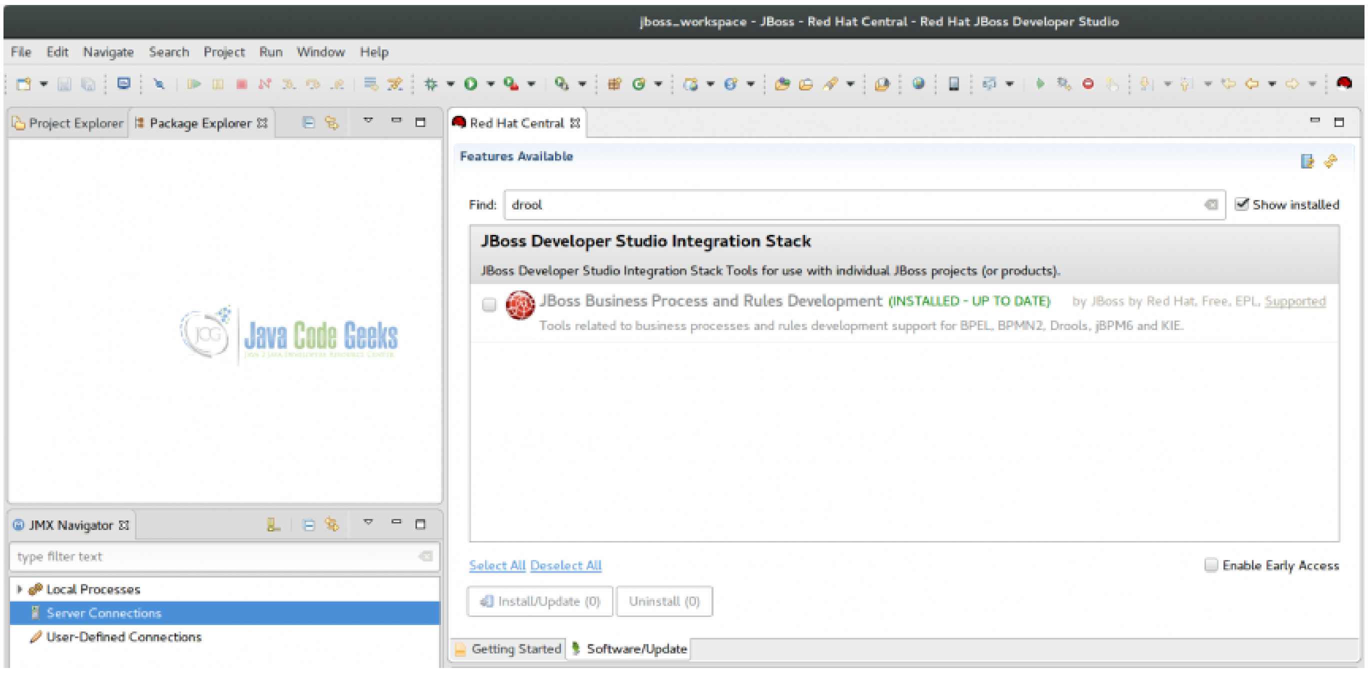
JBoss Data Virtualization Development — by JBoss by Red Hat, Free, EPL, Supported
Tools related to data virtualization such as persistence and transformation - includes support for Teiid Designer.

JBoss Integration and SOA Development — by JBoss by Red Hat, Free, EPL, Supported
Tools related to the development of Integration and SOA applications.

Mobile Development
Plug-ins to support HTML5 Hybrid Mobile application development

Figure 6.2: Jboss plugin page

Use the search box to find the drools plugin and install it.



Find: drool
Show installed: ✓

JBoss Developer Studio Integration Stack
JBoss Developer Studio Integration Stack Tools for use with individual JBoss projects (or products).

JBoss Business Process and Rules Development (INSTALLED - UP TO DATE) — by JBoss by Red Hat, Free, EPL, Supported
Tools related to business processes and rules development support for BPEL, BPMN2, Drools, jBPM6 and KIE.

Figure 6.3: Jboss drools plugin

6.2.1 Creating The Maven Project

After the installation of the drools plugin, we can create a new drool project in order to implement the example.

Go to, file → new → other menu and at the dialog box search for drools project.

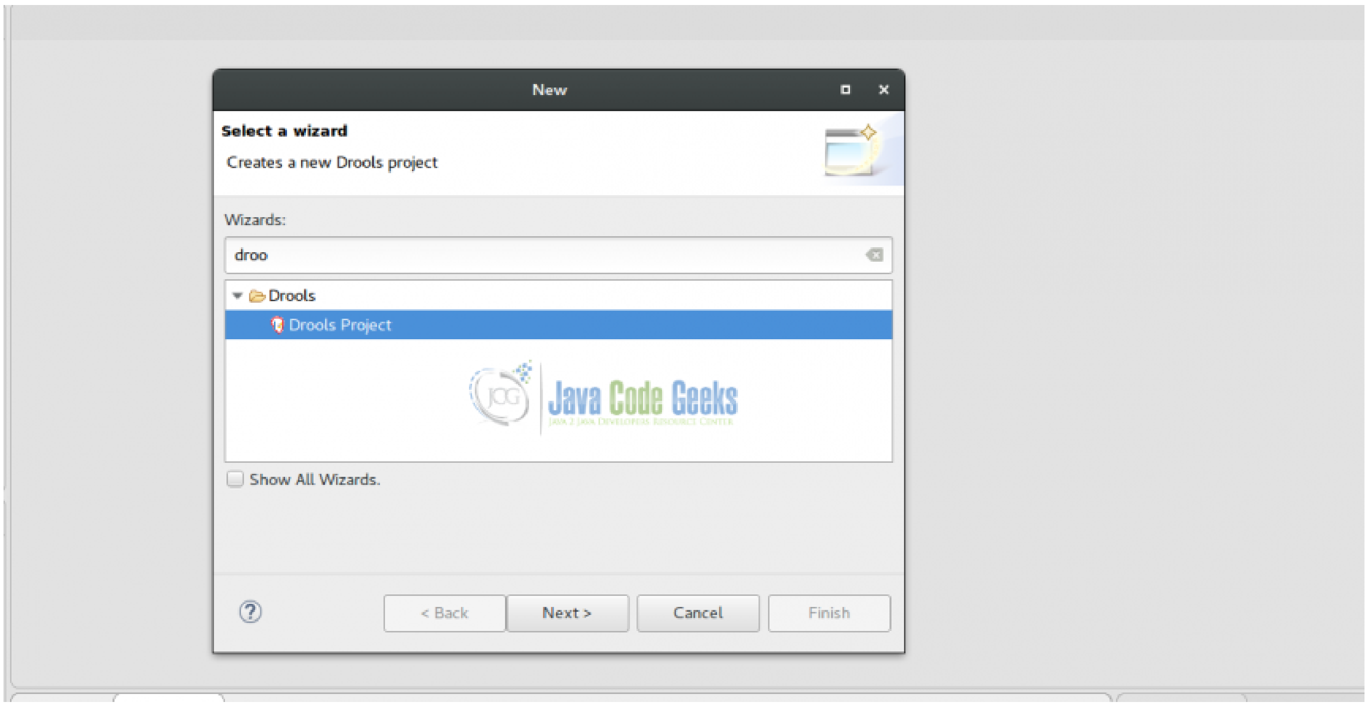
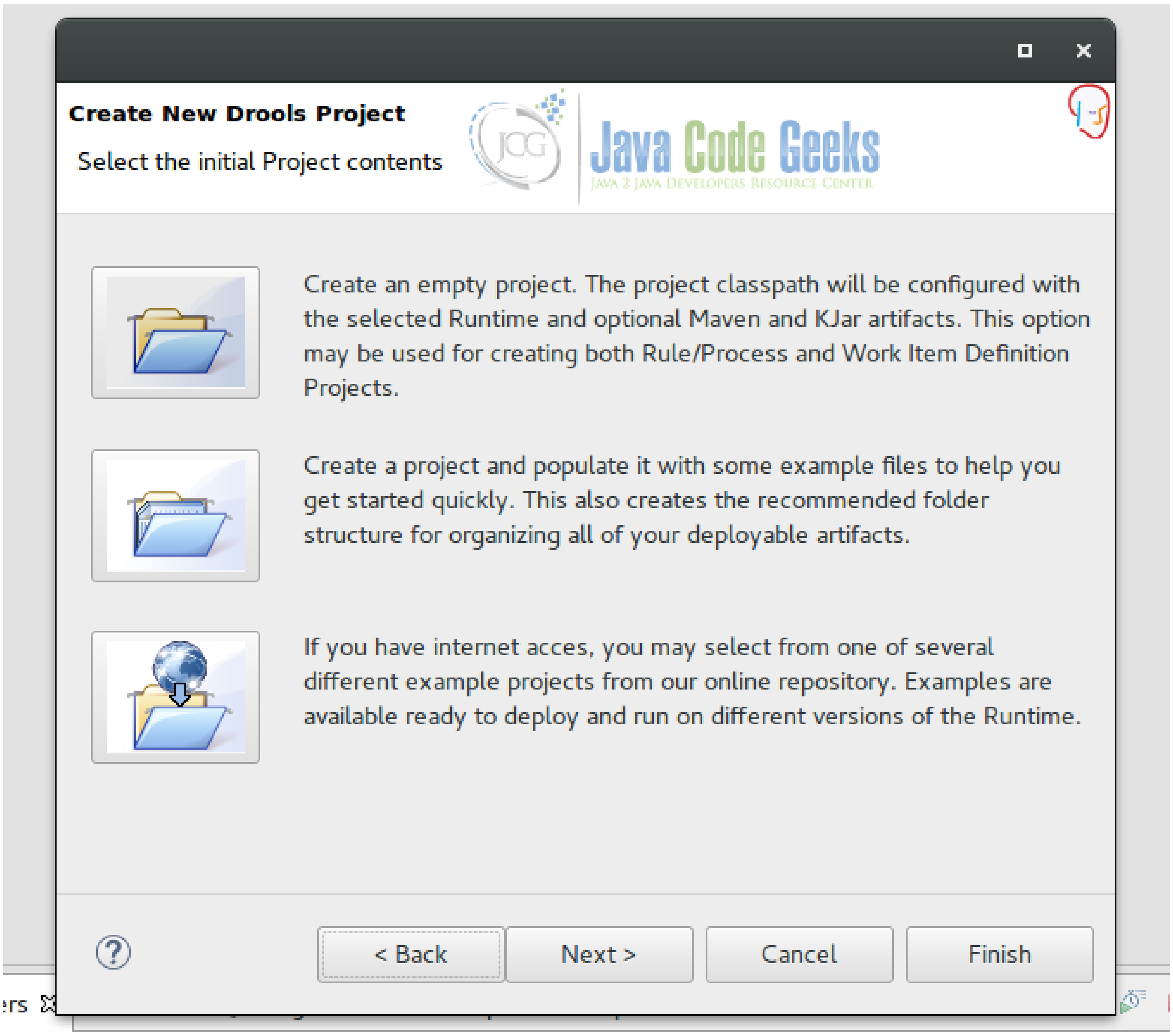


Figure 6.4: Drools project creation

Select the second option, create drools project and populate with some examples.



Create New Drools Project

Select the initial Project contents

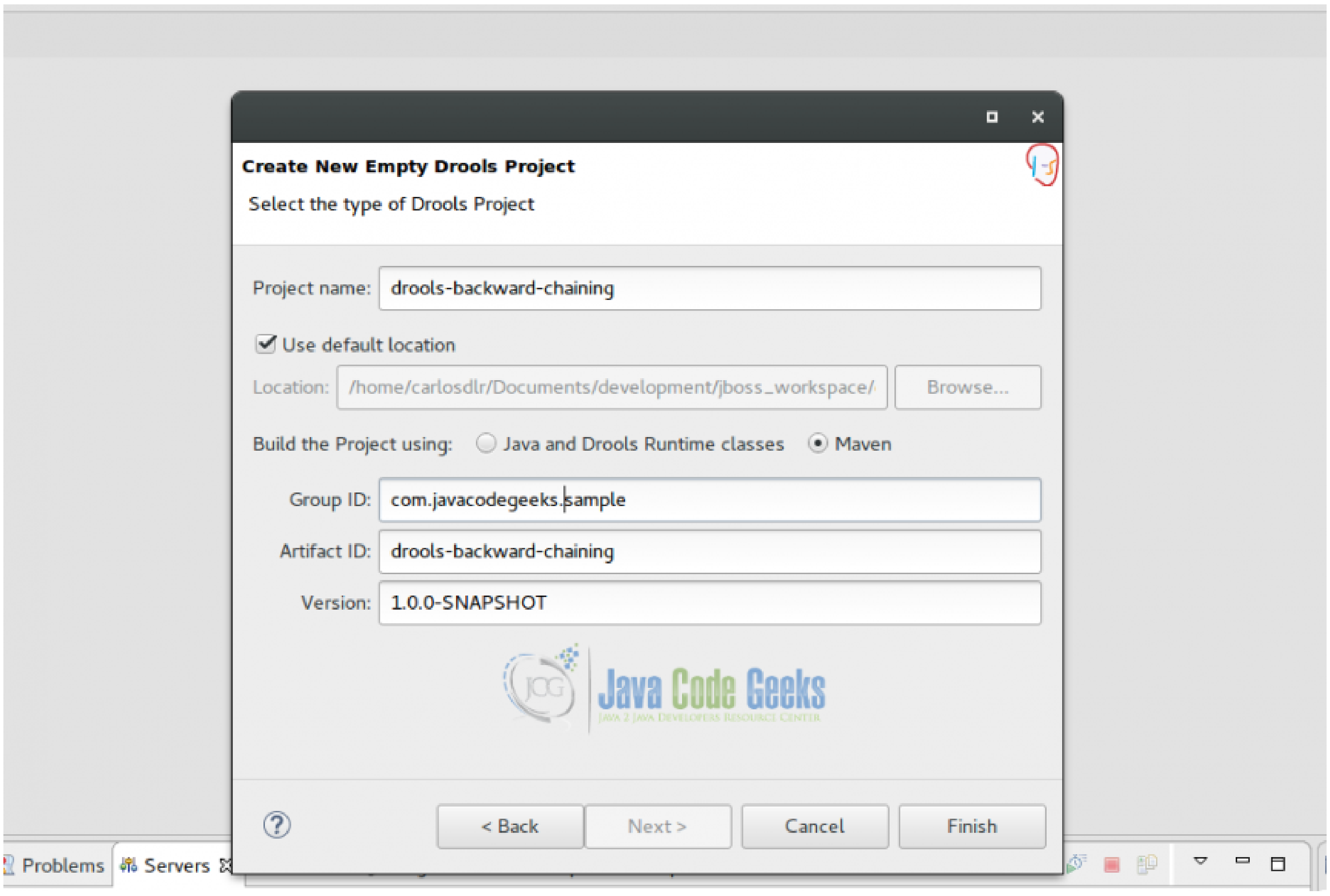
Create an empty project. The project classpath will be configured with the selected Runtime and optional Maven and KJar artifacts. This option may be used for creating both Rule/Process and Work Item Definition Projects.

Create a project and populate it with some example files to help you get started quickly. This also creates the recommended folder structure for organizing all of your deployable artifacts.

If you have internet acces, you may select from one of several different example projects from our online repository. Examples are available ready to deploy and run on different versions of the Runtime.

Figure 6.5: Drools project type

Add a project name and select the maven support.


Create New Empty Drools Project

Select the type of Drools Project

Project name: drools-backward-chaining

[✓] Use default location

Location: /home/carlosdlr/Documents/development/jboss_workspace/

Build the Project using: Java and Drools Runtime classes [● Maven]

Group ID: com.javacodegeeks.sample

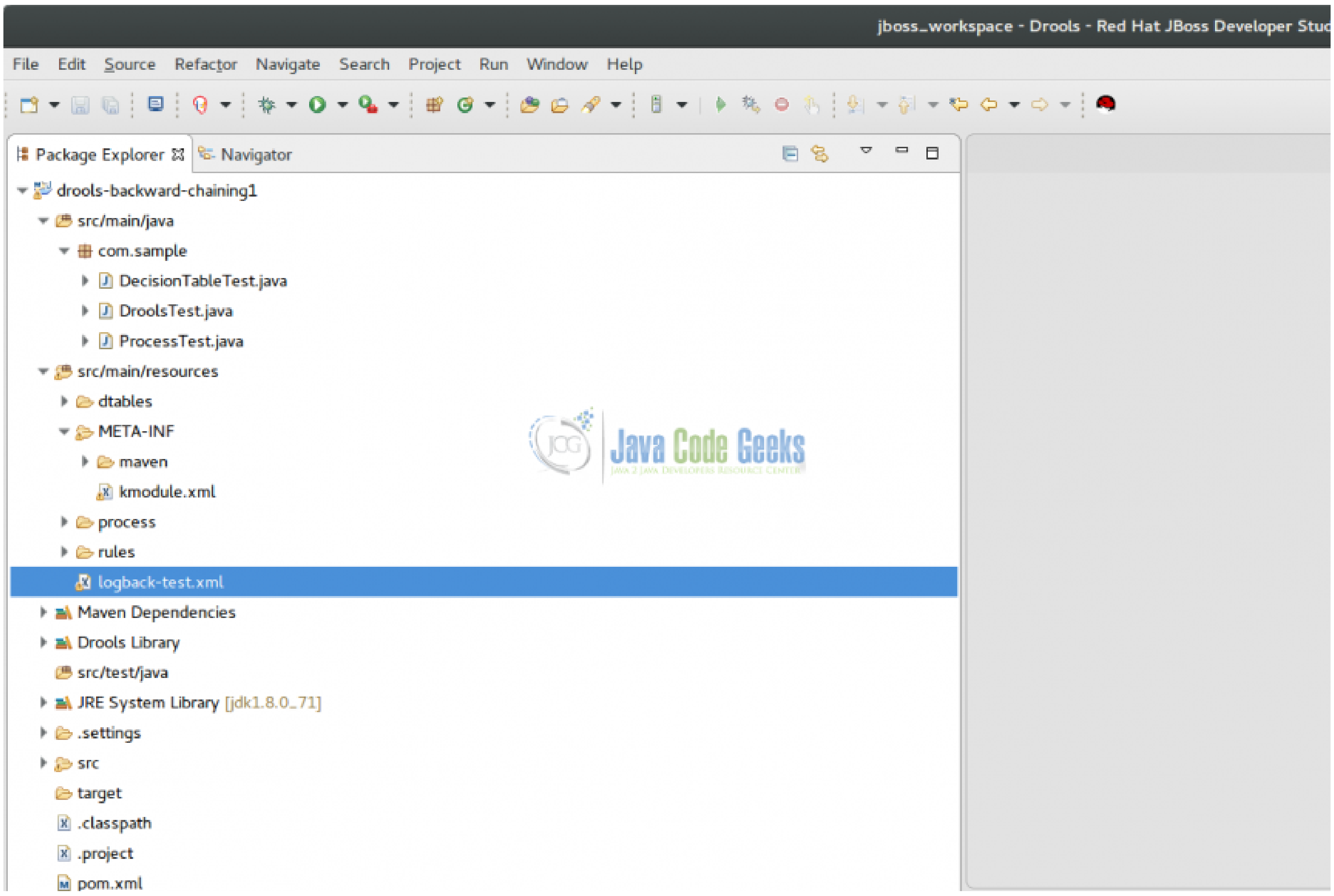
Artifact ID: drools-backward-chaining

Version: 1.0.0-SNAPSHOT

Figure 6.6: Drool project creation

6.2.2 Maven Project Structure

After the project creation wizard, we should have something like this:


jboss_workspace - Drools - Red Hat JBoss Developer Stuc
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Navigator

```
drools-backward-chaining1
  src/main/java
    com.sample
      DecisionTableTest.java
      DroolsTest.java
      ProcessTest.java
  src/main/resources
    dtables
    META-INF
      maven
      kmodule.xml
    process
    rules
    logback-test.xml
  Maven Dependencies
  Drools Library
  src/test/java
  JRE System Library [jdk1.8.0_71]
  .settings
  src
  target
  .classpath
  .project
  pom.xml
```

Figure 6.7: Maven project structure

The project comes with a default package called, com.sample. This has 3 sample classes for a fast getting started and a configuration file named **kmodule.xml** inside **src/main/resources/META-INF** folder, that allows to map our folders that contains the drools rules files (**.drl** extension) in order to create a drools session and execute the rules code.

6.3 Backward Chaining Example Implementation

Now we have already our drools dev environment configured to start the backward chaining implementation. The steps the we will follow are:

- Model Class creation (Location class)
- Drools rules file creation (.drl extension file. Derivation query implementation)
- Drools rule file mapping
- Test Class forresults validation

6.3.1 Model Class Creation

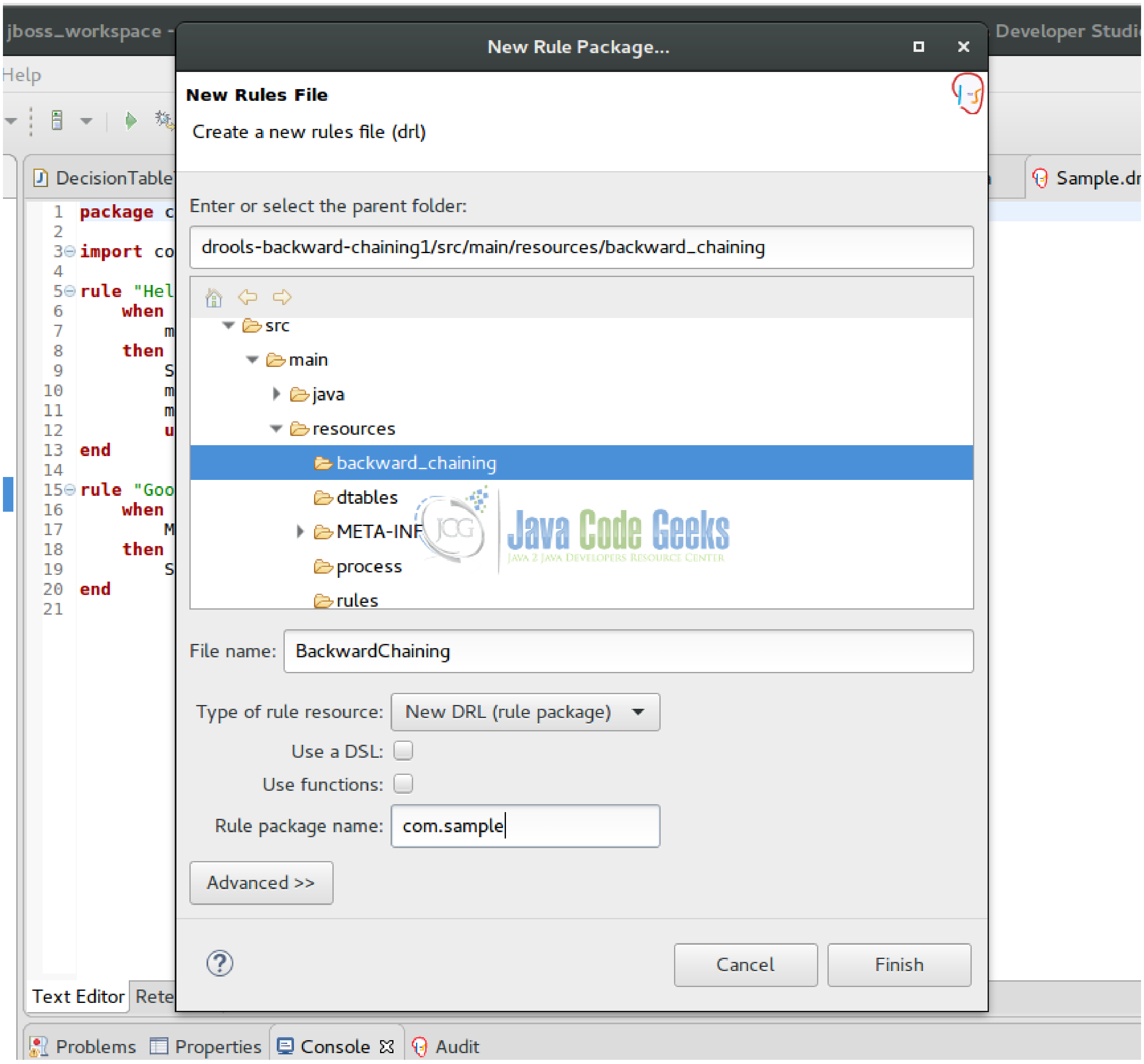
On the jboss studio maven project, right click on **com.sample** package and create a new one **com.sample.model** and inside that package create a new class named Location. This class will be a model to represent the location on the graph.

Location.java

```
package com.sample.model;

import org.kie.api.definition.type.Position;

public class Location {

        @Position(0) //to indicate position of each attribute, that allows to the engine ↩
            identifie the params order to use on the query function
        private String thing;
        @Position(1)
        private String location;

        public Location(){}

        public Location(String thing, String location) {
                super();
                this.thing = thing;
                this.location = location;
        }

        public String getThing() {
                return thing;
        }

        public void setThing(String thing) {
                this.thing = thing;
        }

        public String getLocation() {
                return location;
        }

        public void setLocation(String location) {
                this.location = location;
        }
}
```

6.3.2 Drools Rules File Creation

On the jboss studio maven project, right click on **src/main/resources** folder and create a new folder after called backward_chaining. Right click on the previous folder and create a new rules file named BackwardChaining. Check below how it should look like.


New Rule Package...

New Rules File

Create a new rules file (drl)

Enter or select the parent folder: drools-backward-chaining1/src/main/resources/backward_chaining

File name: BackwardChaining

Type of rule resource: New DRL (rule package)

Use a DSL: ☐

Use functions: ☐

Rule package name: com.sample

Figure 6.8: Rules file creation

BackwardChaining.drl

```
package com.sample

import com.sample.model.Location;
//declare any global variables here
query isContainedIn( String x, String y )
    Location( x, y; ) // we received the initial values from the rule "go1" and start to ↩
        search inside the data stored on the engine
    or
    ( Location( z, y; ) and isContainedIn( x, z; ) ) //recursive call to the function that ↩
        allows to search in a derivation tree structure
end
// rule to print inserted values
```

```
rule "go" salience 10
  when
    $s : String( )
  then
    System.out.println( $s );
end

// rule that invokes the recursive query function to search in our office desk graph and ↩
    when the condition is true prints "key is in the envelop"
rule "go1"

  when
    String( this == "go1" )
    isContainedIn("key", "envelop"; )
  then
    System.out.println( "key is in the envelop" );
end
```

6.3.3 Test Class Creation

On the jboss studio maven project, right click on **com.sample** package and create a new class named **BackwardChainingTest.** This class will be on charge of creating the engine session. Populate the engine with the graph data and invoke the rules.

BackwardChainingTest.java

```
package com.sample;

import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

import com.sample.model.Location;

public class BackwardChainingTest {
        public static void main(String[] args) {
                try {
                        KieServices ks = KieServices.Factory.get();
                        KieContainer kContainer = ks.getKieClasspathContainer();
                        KieSession kSession = kContainer.newKieSession("ksession-backward- ↩
                            chaining");
                         //drools session base on the xml configuration (*kmodule.xml*)
                        //graph population
                        kSession.insert(new Location("desk", "office"));
                        kSession.insert(new Location("flashlight", "desk"));
                        kSession.insert(new Location("envelop", "desk"));
                        kSession.insert(new Location("key", "envelop"));

                        kSession.insert("go1"); //invoke the rule that calls the query  ↩
                            implentation of backward chaining
                        kSession.fireAllRules(); //invoke all the rules
                } catch (Throwable t) {
                        t.printStackTrace();
                }
        }

}
```

The output of the previous program is:

```
"go1
key is in the envelop"
```

This feature added to the drools engine "Backward chaining" is goal driven. That means, that the rule system has a condition that tries to satisfy. If this is not possible search for other possibles values and continue this operation until the condition satisfied.

6.4 Conclusions

On this example we saw how to create the backward chaining implementation and use it on a project example using jboss dev studio. Also we understood how the data is stored inside the drools engine and how to analyze this, by using graphs in order to get a more clear view.

If you want to read more about this feature please go: **Backward Chaining**

6.5 Download the Eclipse Project

This was a Drools backward chaining Example with Eclipse

Download

You can download the full source code of this example here: **drools-backward-chaining**

Chapter 7

Jboss Drools AgendaEventListener Example

Hello Readers, in this article we will take a look on how to use and implement an AgendaEventListener on a drools rule program. Before we start, check the requirements related to technologies and frameworks used to this example below:

- Maven 3.3.9
- Jboss Studio 10.3
- Drools Engine 7.0
- JDK 1.8.0_71

7.1 Introduction

Before continuing with the AgendaEventListener Example implementation, we need to understand how this works and which is the use of this feature inside a drools rule program.

7.1.1 Event Model

The drools API provides a package called org.kie.api.event. This package has all the classes that will use to add event listeners inside a rule program that will be covered on this example. The event drools's API, provides a way to notify the user when a rule event is triggered. This includes assertion on objects values, conditions, etc.

Therefore, this API allows to separate logging and auditing tasks from your main rule program or add other functionalities, using a callback strategy.

7.1.2 Checking The API

In order to get more understanding on how the API works, we'll check out the **UML** diagram for a rule event manager, that we will covered on this example.

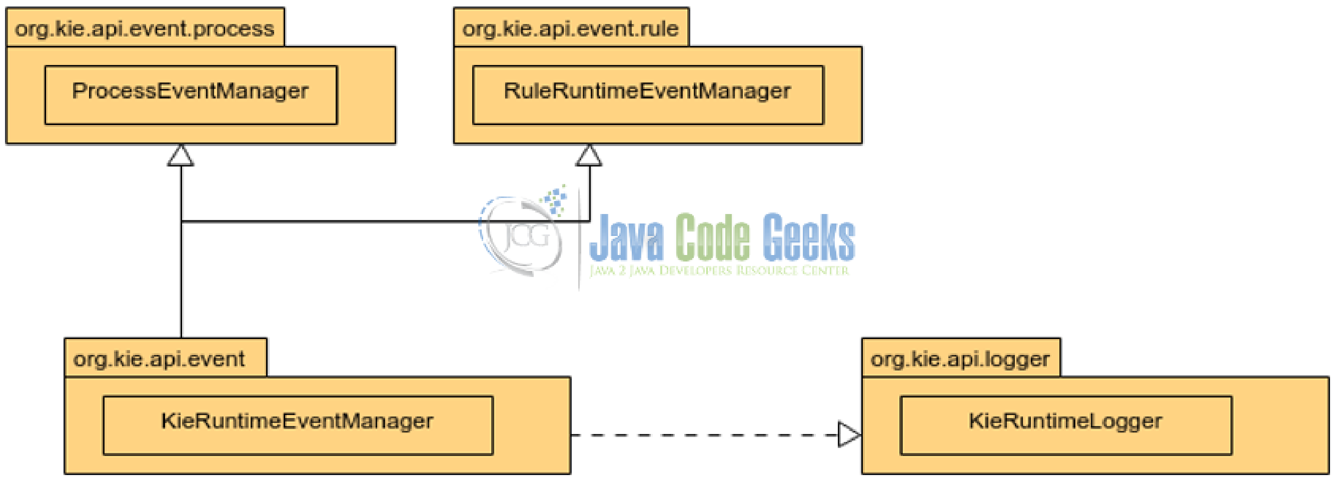


Figure 7.1: KieRuntimeEventManager

The KieRuntimeManager interface is implemented by KieRuntime and provides other 2 interfaces RuleRuntimeEventManager this interface is used by the rules programs and ProcessEventManager is used by the BPM process model programs to add the listener feature that allows add more functionality to our drools programs. On this example we will cover the RuleRuntimeEventManager API.

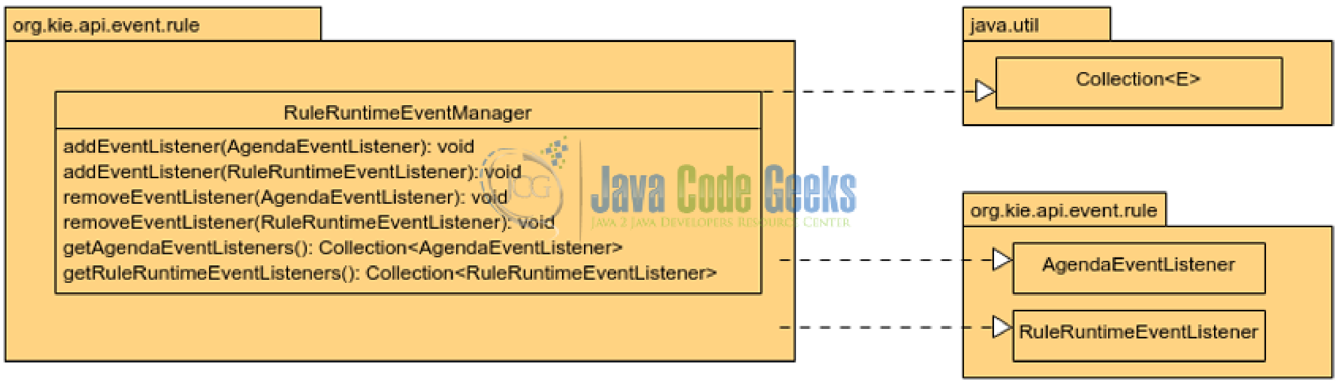

org.kie.api.event.rule
RuleRuntimeEventManager
addEventListener(AgendaEventListener): void
addEventListener(RuleRuntimeEventListener): void
removeEventListener(AgendaEventListener): void
removeEventListener(RuleRuntimeEventListener): void
getAgendaEventListeners(): Collection<AgendaEventListener>
getRuleRuntimeEventListeners(): Collection<RuleRuntimeEventListener>
java.util
Collection<E>
org.kie.api.event.rule
AgendaEventListener
RuleRuntimeEventListener

Figure 7.2: RuleRuntimeEventManager

The RuleRuntimeEventManager API expose the necessary methods to add or remove listeners, so the events related to the working memory inside a rule workflow can be listened to.

Additionally, this API expose some other methods that allows to get information about of which listeners has our rule program.

7.2 Setting Up The Environment

To have the dev environment setting up, please refer to my previous drools post (backward chaining) on the section 2 "Configure Necessary Tools" and uses the name `drools-agenda-event-listener` for the new maven drools project used on this example.

7.3 AgendaEventListener Example Implementation

Well, now in this section, we'll start to implement our first agendaEventListener on a drools rule project. Below we see the steps that we'll follow to achieve this.

- Model creation class to wrap the data that will be evaluated by the rule.
- Rule file with some example rule, in order to add a new agendaEventListener.
- Add rule configuration to the `kmodule.xml` file, in order to get our rule on a drools session.
- Test class, to put all together and see how the AgendaEventListener works.

7.3.1 Model Class Creation

The model class is a representation of the data that will be evaluated by the rule. On this example we will use a class called message. To create it please follow the next steps. Go to the `drools-agenda-event-listener` (created on the step 2) maven project on the jboss developer studio and create a new package named `com.sample.model`. Inside this package create a new class named `Message` with the below structure:

`Message.java`

```
package com.sample.model;

/**
 * POJO class to wrap the example data
 * that will be evaluated by the rule
 *
 */
public class Message {

        private int code;
        private String text;

        public Message(int code, String text) {
                super();
                this.code = code;
                this.text = text;
        }

        public int getCode() {
                return code;
        }

        public void setCode(int code) {
                this.code = code;
        }

        public String getText() {
                return text;
        }

        public void setText(String text) {
                this.text = text;
        }
}
```

7.3.2 Rule File Creation

The rule file will have our test rule that allows validate the data model when this is invoked and will allow to trigger our event listener. On the `drools-agenda-event-listener` maven project, inside the src/main/resources create a new folder named `agendaeventlistenerrule` . Below this folder create a new Rule file named `AgendaEventListenerSample` with the below structure:

`AgendaEventListenerSample.drl`

```
package com.sample

import com.sample.model.Message


/**
When this rule matched, the agenda event AfterMatchFiredEvent will be fired
*/
rule "agendatest"

    when
        Message( code == 20, text == "agenda example rule" )
    then
         System.out.println( "Rule output" );

end
```

7.3.3 Add Rule Configuration

In order to get our rule program working and see how the events listeners works, it's necessary to configure the drools session on the file `kmodule.xml` file inside src/main/resources/META-INF folder. See below the configuration for this example:

`kmodule.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns="https://jboss.org/kie/6.0.0/kmodule">

    <!-- drools event listener session mapping -->
     <kbase name="agendaeventlistenerrule" packages="agendaeventlistenerrule">
        <ksession name="ksession-agendaeventlistenerrule"/>
    </kbase>
</kmodule>
```

7.3.4 Test Class Creation

Now, we are ready to add our new AgendaEventListener implementation and test it. On the `drools-agenda-event-listener` maven project, under the package `com.sample` create a new class named `AgendaEventListenerTest` with the below structure:

`AgendaEventListenerTest.java`

```
package com.sample;

import org.kie.api.KieServices;
import org.kie.api.event.rule.AfterMatchFiredEvent;
import org.kie.api.event.rule.DefaultAgendaEventListener;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import com.sample.model.Message;

public class AgendaEventListenerTest {

        public static void main(String[] args) {
                try {
                        KieServices ks = KieServices.Factory.get();
                        KieContainer kContainer = ks.getKieClasspathContainer();

                        //drools session base on the xml configuration kmodule.xml
                        KieSession kSession = kContainer.newKieSession("ksession- ↩
                            agendaeventlistenerrule");

                        /*add the event listener in this case we'll use the  ↩
                            DefaultAgendaEventListener,
                        that is a implementation of AgendaEventListener*/
                        kSession.addEventListener(new DefaultAgendaEventListener() {

                                //this event will be executed after the rule matches with  ↩
                                    the model data
                                public void afterMatchFired(AfterMatchFiredEvent event) {
                                super.afterMatchFired(event);
                                System.out.println(event.getMatch().getRule().getName());// ↩
                                    prints the rule name that fires the event
                            }
                        });

                        //add the model with the data that will match with the rule  ↩
                            condition
                        kSession.insert(new Message(20, "agenda example rule"));

                        //fire all the rules
                        kSession.fireAllRules();
                } catch (Throwable t) {
                        t.printStackTrace();
                }
        }
}
```

The output of this program is:

```
Rule output // This is printing when the rule matches
agendatest // This is printing by the listener after the rule matches
```

7.4 Conclusion

On this example, we learned how to make a simple implementation of listeners inside a rule workflow, how the event API is designed inside drools engine and how to use it on a drools rule program.

This drools engine feature allow us to add monitoring or logging on a rules projects, using the callback approach in order to get a clean way to add more functionality to our drools rule projects.

If you want to read more about this feature please go: **AgendaEventListener**

7.5 Download the Eclipse Project

This was a Drools AgentEventListener example with Jboss developer studio.

Download

You can download the full source code of this example here: **drools-agenda-event-listener**