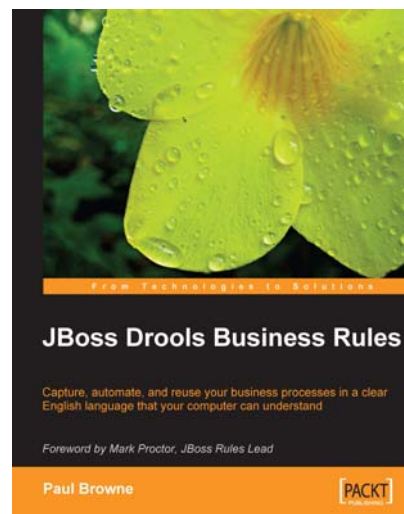




# JBoss Drools Business Rules

**Paul Browne**



## Chapter No. 4 "Guided Rules with the Guvnor"

## In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.4 "Guided Rules with the Guvnor"

A synopsis of the book's content

Information on where to buy this book

## About the Author

**Paul Browne's** first job was selling computers in France and things went steadily downhill from there. He spent millions on behalf of a UK telephone company's procurement department and implemented direct marketing for a well-known Texan computer maker before joining the IT department of a company that builds bright red tractors and other seriously cool machines.

Paul then embraced his techie side (he was writing games in machine code from the age of 11) and started a consultancy that used IT to solve business problems for companies in the financial and public sectors in Ireland, UK, Belgium, and New Zealand. Eight years later, he now works with an Irish government agency that helps similar software companies to grow past their initial teething pains.

More formally, Paul has a bachelor's degree in Business and French from the University of Ulster, a master's degree in Advanced Software from UCD Dublin, a post-grad qualification in Procurement from the Chartered Institute of Procurement and Supply (UK), and will someday complete his ACCA financial exams.

Paul can be found on LinkedIn at <http://www.linkedin.com/in/paulbrowne>, and via the Red Piranha (Business knowledge) project at <http://code.google.com/p/red-piranha/>.

---

I would like to thank my parents for the gift of learning; my wife and family for the constant encouragement to write this book; the work colleagues that I've had the pleasure to learn from and all the people behind the Drools and other outstanding open software projects.

---

**For More Information:** [www.packtpub.com/jboss-drools-business-rules/book](http://www.packtpub.com/jboss-drools-business-rules/book)

# JBoss Drools Business Rules

In business, a lot of actions are triggered by rules: "Order more ice cream when the stock is below 100 units and temperature is above 25° C", "Approve credit card application when the credit background check is OK, past relationship with the customer is profitable, and identity is confirmed", and so on. Traditional computer programming languages make it difficult to translate this "natural language" into a software program. But JBoss Rules (also known as Drools) enables anybody with basic IT skills and an understanding of the business to turn statements such as these into running computer code.

This book will teach you to specify business rules using JBoss Drools, and then put them into action in your business. You will be able to create rules that trigger actions and decisions, based on data that comes from a variety of sources and departments right across your business. Regardless of the size of your business, you can make your processes more effective and manageable by adopting JBoss Rules.

Banks use business rules to process your mortgage (home loan) application, and to manage the process through each step (initial indication of amount available, actual application, approval of the total according to strict rules regarding the amount of income, house value, previous repayment record, swapping title deeds, and so on).

Countries such as Australia apply business rules to visa applications (when you want to go and live there)—you get points for your age, whether you have a degree or masters, your occupation, any family members in the country, and a variety of other factors.

Supermarkets apply business rules to what stock they should have on their shelves and where—this depends upon analyzing factors such as how much shelf space there is, what location the supermarket is in, what people have bought the week before, the weather forecast for next week (for example, ice cream in hot weather), and what discounts the manufacturers are giving.

This book shows how you can use similar rules and processes in your business or organization. It begins with a detailed, clear explanation of business rules and how JBoss Rules supports them.

You will then see how to install and get to grips with the essential software required to use JBoss Rules. Once you have mastered the basic tools, you will learn how to build practical and effective of the business rule systems.

The book provides clear explanations of business rule jargon. You will learn how to work with Decision Tables, **Domain-Specific Languages (DSL)s**, the Guvnor and JBoss **Integrated Development Environment (IDE)**, workflow and much more.

By the end of the book you will know exactly how to harness the power of JBoss Rules in your business.

For More Information: [www.packtpub.com/jboss-drools-business-rules/book](http://www.packtpub.com/jboss-drools-business-rules/book)

## What This Book Covers

*Chapter 1:* This chapter gives you a good platform to understand business rules and JBoss rules. We look at the problems that you might have (and why you're probably reading this book). We look at what business rule engines are, and how they evaluate business rules that appear very simple and how they become powerful when multiple rules are combined.

*Chapter 2:* This chapter explains setting up Java, setting up **Business Rule Management System (BRMS)**/Guvnor running on the JBoss App Server, setting up Eclipse, and installing the Drools Plug-in. It also details the installation of the Drools examples for this book and the Maven to build them.

*Chapter 3:* Guvnor is the user-friendly web editor that's also powerful enough to test our rules as we write them. We take up an example to make things easier. Then we look at the various Guvnor screens, and see that it can not only write rules (using both guided and advanced editors), but that it can also organize rules and other assets in packages, and also allow us to test and deploy those packages. Finally, we write our very first business rule—the traditional 'Hello World' message announcing to everyone that we are now business rule authors.

*Chapter 4:* This chapter shows how to use the Guvnor rule editor to write some more sophisticated rules. It also shows how to get information in and out of our rules, and demonstrates how to create the fact model needed to do this. We import our new fact model into the Guvnor and then build a guided rule around it. Finally we test our rule as a way of making sure that it runs correctly.

*Chapter 5:* This chapter pushes the boundaries of what we can do with the Guvnor rule editor, and then brings in the JBoss IDE as an even more powerful way of writing rules. We start by using variables in our rules example. Then we discuss rule attributes (such as salience) to stop our rules from making changes that cause them to fire again and again. After testing this successfully, we look at text-based rules, in both the Guvnor and the JBoss IDE, for running 'Hello World' in the new environment.

*Chapter 6:* This chapter looks again at the structure of a rule file. At the end of this chapter, we look at some more advanced rules that we can write and run in the IDE.

*Chapter 7:* This chapter explains how testing is not a standalone activity, but part of an ongoing cycle. In this chapter we see how to test our rules, not only in the Guvnor, but also using FIT for rule testing against requirements documents. This chapter also explains Unit Testing using JUnit.

For More Information: [www.packtpub.com/jboss-drools-business-rules/book](http://www.packtpub.com/jboss-drools-business-rules/book)

*Chapter 8:* This chapter explains how to use Excel Spreadsheets (cells and ranges) as our fact model to hold information, instead of the write-your-own-JavaBean approach we took earlier. Then we use Excel spreadsheets to hold Decision tables, to make repetitive rules easier to write.

*Chapter 9:* This chapter aims to make our rules both easier to use, and more powerful. We start with DSLs—Domain-Specific Languages. This chapter follows on from the 'easy to write rules' theme from the previous chapter and also discusses both rule flow and workflow.. It would be great to draw a workflow diagram to see/control what (groups of) rules should fire and when. Rule flow gives us this sort of control.

*Chapter 10:* This chapter shows you how to deploy your business rules into the real world. We look at the pieces that make up an entire web application, and where rules fit into it. We see the various options to deploy rules as part of our application, and the team involved in doing so. Once they are deployed, we look at the code that would load and run the rules—both home-grown and using the standard RuleAgent. Finally we see how to combine this into a web project using the framework of your choice.

*Chapter 11:* This chapter looks at what happens under the cover by opening up the internals of the Drools rule engine to understand concepts such as truth maintenance, conflict resolution, pattern matching, and the rules agenda. In this chapter, we explore the Rete algorithm and discuss why it makes rules run faster than most comparable business logic. Finally we see the working memory audit log and the rules debug capabilities of the Drools IDE.

*Chapter 12:* This chapter deals with the other advanced Drools features that have not yet been covered. This includes Smooks to bulk load data, Complex Event Processing, and Drools solver to provide solutions where traditional techniques would take too long.


# 4

## Guided Rules with the Guvnor

In the last chapter we took a tour with the Guvnor and used it to write our first business rule, and printed out a traditional 'Hello World' message. Although this rule is a major step forward for us, we're not really using the full power of the Drools rule engine. In this chapter, we're going to stay with the Guvnor rule editor, and use it to write some more sophisticated rules. In particular, we're going to:

- Show how to put information into and out of our rules
- Build a fact model to hold this information
- Import our newly built model into Guvnor
- Create guided rules using this fact model
- Run and test our new fact-based rules

### Passing information in and out

The main reason for the simplicity of our Hello World example was that it neither took in any information, nor passed any information out – the rule always fired, and said the same thing. In real life, we need to pass information between our rules and the rest of the system. You may remember that in our tour of the Guvnor, we came across models that solved this problem of 'How do we get information into and out of the rules?'.  
[  If you're familiar with Java, models are just normal JavaBeans deployed into Guvnor/JBoss rules in a JAR (ZIP-like) file; nothing more, nothing less. In fact, a lot of the time you can use the JavaBeans that already exist in your system. ]

For More Information: [www.packtpub.com/jboss-drools-business-rules/book](http://www.packtpub.com/jboss-drools-business-rules/book)

Here's a quick reminder of the spreadsheet that we used as an example in the last chapter:

	A	B	C	D
1	<b>Customer Name</b>	<b>Sales</b>	<b>Date of Sale</b>	<b>Chocolate Only Customer</b>
2	Acme Corp	\$100,000	01-Feb	Y
3	Breakfast Roll Inc	\$250,000	01-Mar	N
4	Chocolate Creams Co.	\$30,000	01-Apr	Y
5	Dunkin Dreams	\$200,000	01-May	Y
6	Easy Eating	\$150,000	01-Jun	N

If we want to duplicate this in our model/JavaBean, we would need places to hold four key bits of sales-related information.

- Customer Name: String (that is, a bit of text)
- Sales: Number
- Date of Sale: Date
- Chocolate Only Customer: Boolean (that is, a Y/N type field)

We also need a description for this group of information that is useful when we have many spreadsheets/models in our system (similar to the way this spreadsheet tab is called **Sales**)

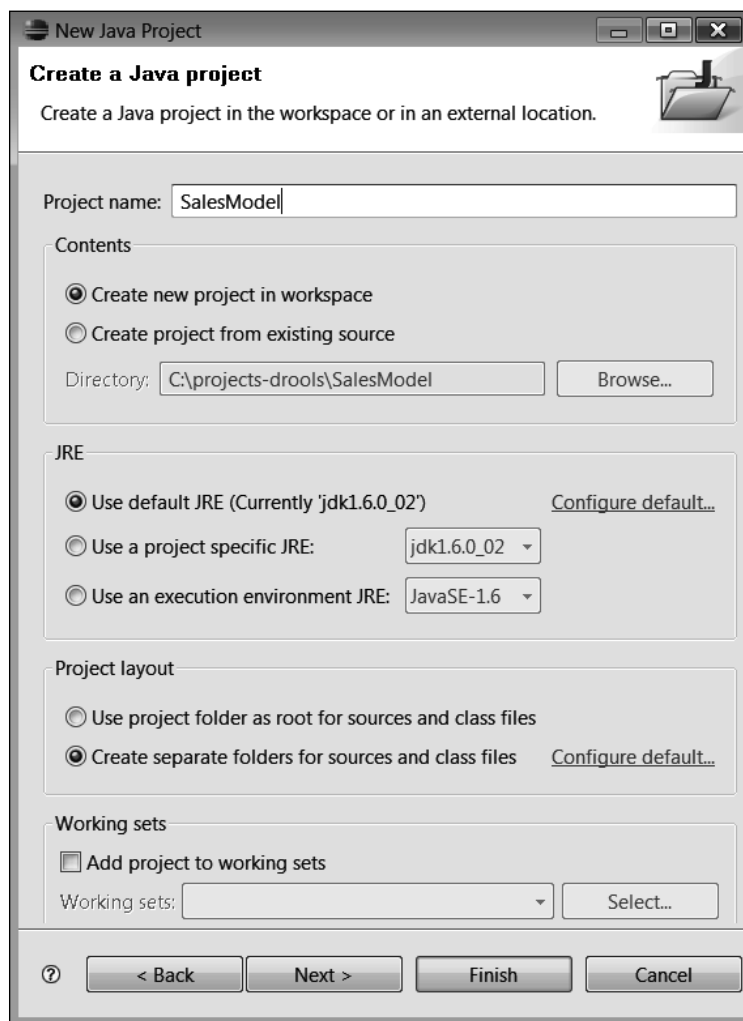


Note that one JavaBean (model) is equal to one line in the spreadsheet. Because we can have multiple copies of JavaBeans in memory, we are able to represent the many lines of information that we have in a spreadsheet. Later, we'll loop and add 10, 100, or 1000 lines (that is, JavaBeans) of information into Drools (for as many lines as we need). As we loop, adding them one at a time, the various rules will fire as a match is made.

## Building the fact model

We will now build this model in Java using the Eclipse editor we installed in Chapter 2. Don't worry if this is your first bit of Java; we're going to do it step-by-step.

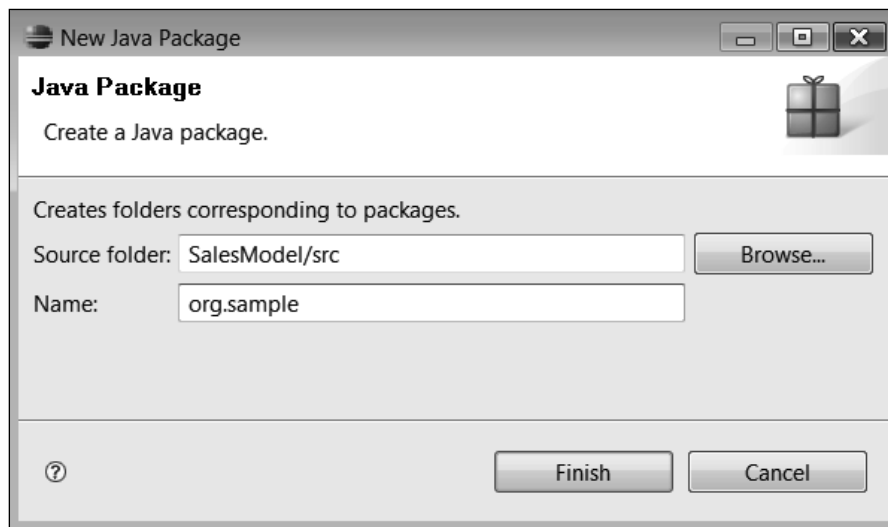
1. Open the Eclipse/JBoss IDE editor that you installed earlier. If prompted, use the default workspace. (Unless you've a good reason to put it somewhere else.)
2. From the menu bar at the top the screen, select **File | New Project**. Then choose **Java Project** from the dialog box that appears. You can either select this by starting to type "Java Project" into the wizard, or by finding it by expanding the various menus.
3. In the **Create a new Java Project** dialog that appears, give the project a name in the upper box. For our example, we'll call it **SalesModel** (one word, no spaces).
4. Accept the other defaults (unless you have any other reason to change them). Our screen will now look something like this:





When you've finished entering the details, click on **Finish**. You will be redirected to the main screen, with a new project (**SalesModel**) created. If you can't see the project, try opening either the **Package** or the **Navigator** tab.

When you can see the project name, right-click on it. From the menu, choose **New | Package**. The **New Java Package** dialog will be displayed, as shown below. Enter the details as per the screenshot to create a new package called **org.sample**, and then click on **Finish**.



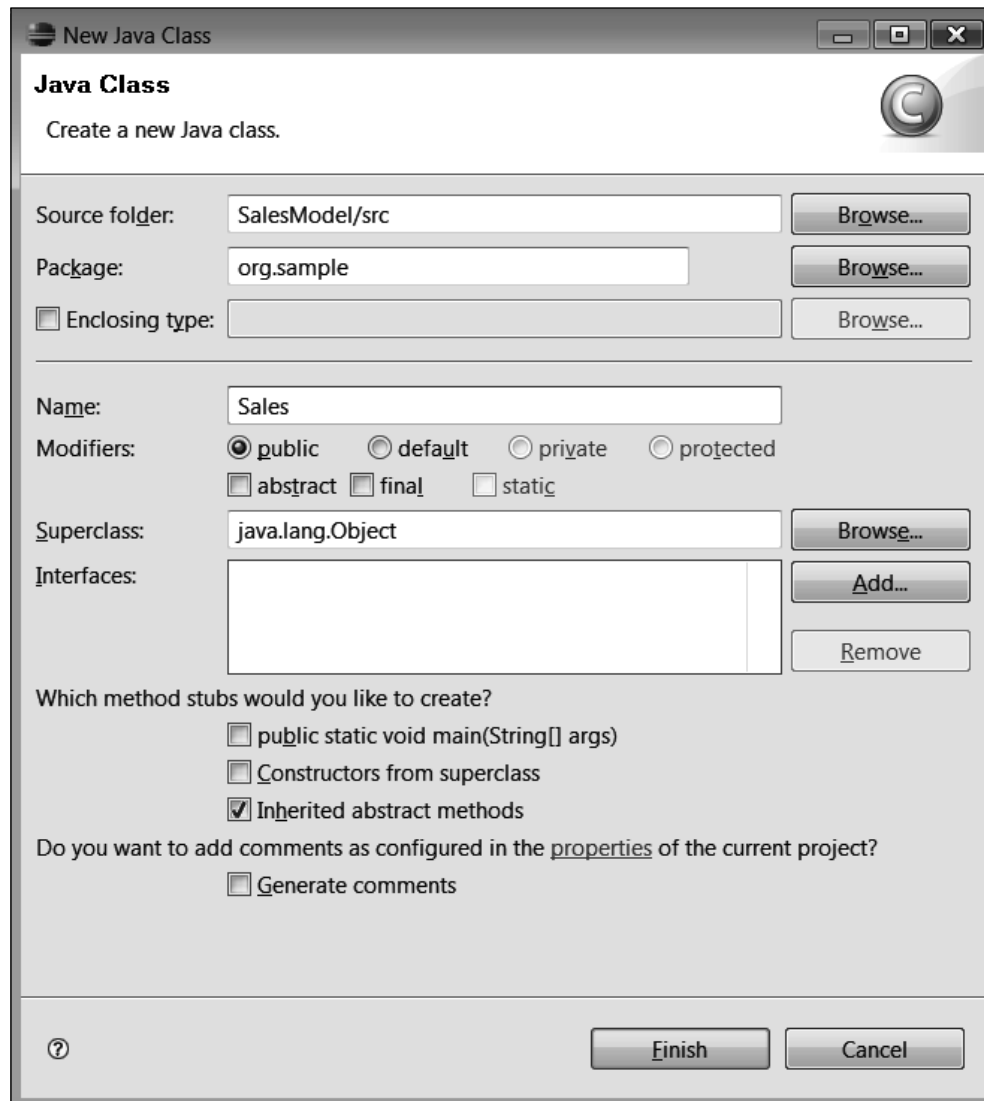
If you are doing this via the navigator (or you can take a peek via Windows Explorer), you'll see that this creates a new folder **org**, and within it a subfolder called **sample**. Now that we've created a set of folders to organize our JavaBeans, let's create the JavaBean itself by creating a class.



Did you play with Lego blocks as a kid – multicolored plastic blocks that you could pull apart and stick together again and again? JavaBeans are like those Lego blocks – instead of building toy houses, we can build entire computer systems with them.

Often, while playing Lego, you'd run out of blocks (often red roof tiles) just when you were about to finish. Luckily, in Java, we can create as many blocks as we want. The class that we're about to put together is our mould to let us do this.

To create a new Java class, expand/select the `org.sample` package (folder) that we created in the previous step. Right-click on it and select **New Class**. Fill in the dialog as shown in the following screenshot, and then click on **Finish**:



We will now be back in the main editor, with a newly created class called `Sales.java` (below). For the moment, there isn't much there—it's akin to two nested folders (a sample folder within one called `org`) and a new (but almost empty) file / spreadsheet called `Sales`.

```
package org.sample;
public class Sales {
}
```

By itself, this is not of much use. We need to tell Java about the information that we want our class (and hence the beans that it creates) to hold. This is similar to adding new columns to a spreadsheet.

Edit the Java class until it looks something like the code that follows (and take a quick look of the notes information box further down the page if you want to save a bit of typing). If you do it correctly, you should have no red marks on the editor (the red marks look a little like the spell checking in Microsoft Word).

```
package org.sample;
import java.util.Date;
public class Sales {
    private String name;
    private long sales;
    private Date dateOfSale;
    private boolean chocolateOnlyCustomer;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public long getSales() {
        return sales;
    }

    public void setSales(long sales) {
        this.sales = sales;
    }

    public Date getDateOfSale() {
        return dateOfSale;
    }

    public void setDateOfSale(Date dateOfSale) {
        this.dateOfSale = dateOfSale;
    }
}
```

```

    }

    public boolean isChocolateOnlyCustomer() {
        return chocolateOnlyCustomer;
    }

    public void setChocolateOnlyCustomer(boolean chocolateOnlyCustomer) {
        this.chocolateOnlyCustomer = chocolateOnlyCustomer;
    }
}

```

Believe it or not, this piece of Java code is almost the same as the Excel Spreadsheet we saw at the beginning of the chapter. If you want the exact details, let's go through what it means line by line.

- The braces ({ and }) are a bit like tabs. We use them to organize our code.
- `package`—This data holder will live in the subdirectory `sample` within the directory `org`.
- `import`—List of any other data formats that we need (for example, dates). Text and number data formats are automatically imported.
- `Public class Sales`—This is the mould that we'll use to create a JavaBean. It's equivalent to a spreadsheet with a **Sales** tab.
- `Private String name`—create a text (string) field and give it a column heading of 'name'. The private bit means 'keep it hidden for the moment'.
- The next three lines do the same thing, but for `sales` (as a number/long), `dateOfSale` (as a date) and `chocolateOnlyCustomer` (a Boolean or Y/N field).
- The rest of the lines (for example, `getName` and `setName`) are how we control access to our private hidden fields. If you look closely, they follow a similar naming pattern.



The `get` and `set` lines (in the previous code) are known as **accessor methods**. They control access to hidden or private fields. They're more complicated than may seem necessary for our simple example, as Java has a lot more power than we're using at the moment.

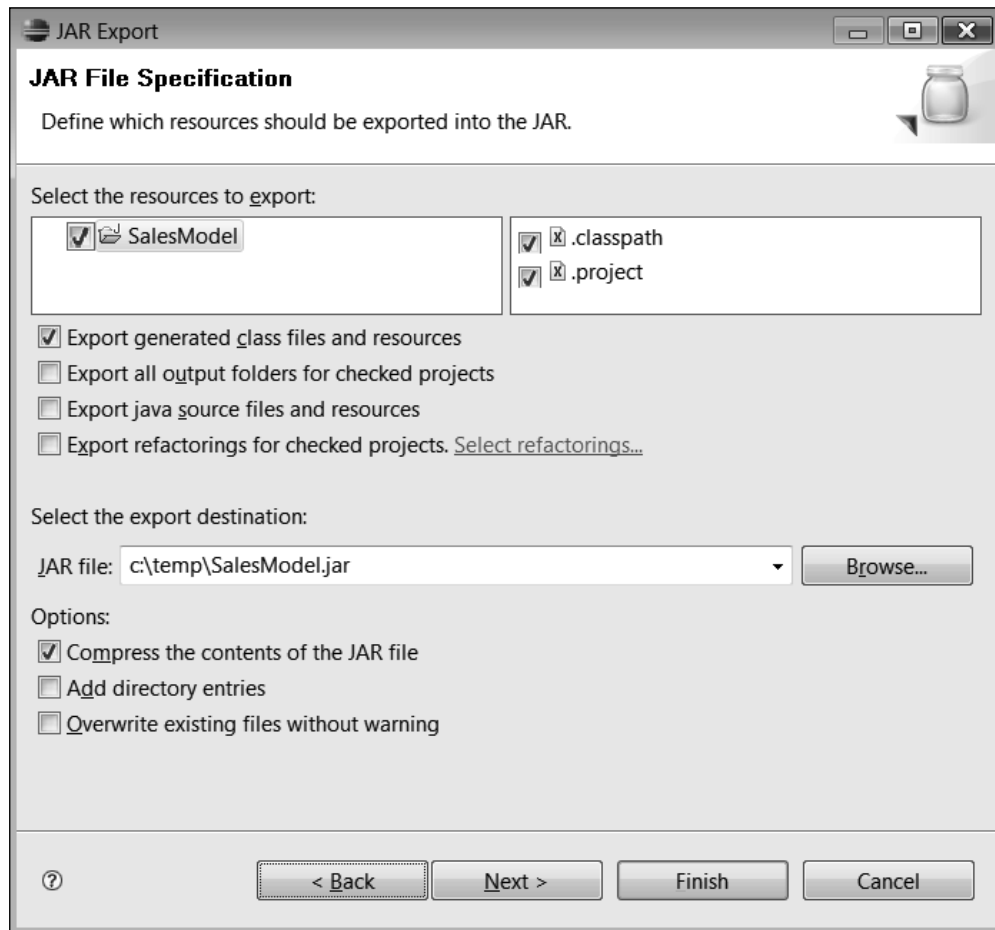
Luckily, Eclipse can auto-generate these for us. (Right-click on the word **Sales** in the editor, then select **Source | Generate Getters and Setters** from the context menu. You should be prompted for the Accessor methods that you wish to create.)

Once you have the text edited like the sample above, check again that there are no spelling mistakes. A quick way to do this is to check the **Problems** tab in Eclipse (which is normally at the bottom of the screen).



If you do have any problems, you may be able to use the Eclipse quick-fix feature (highlight the problem, then press *Ctrl+1*). If that doesn't work, check again and ensure that the spelling is exactly the same as shown earlier. If that doesn't work, follow the steps in the *How to ask for help* section near the beginning of this book.

Now that we've created our model in Java, we need to export it so that we can use in the Guvnor.



1. In Eclipse, right-click on the project name (**SalesModel**) and select **Export**.
2. From the pop-up menu, select **jar** (this may be under Java; you might need to type **jar** to bring it up). Click on **Next**. The screen shown above will be displayed.
3. Fill out this screen. Accept the defaults, but give the JAR file a name (**SalesModel.jar**) and a location (in our case **C:\temp\SalesModel.jar**). Remember these settings as we'll need them shortly.
4. All being well, you should get an 'export successful' message when you click on the **Finish** button, and you will be able to use Windows Explorer to find the JAR file that you just created.



What is a JAR file? JAR stands for Java Archive and is just another name for a ZIP compressed file (you may be familiar with the WinZip utility). Although our model is pretty small (only one file), compressing the files and putting them in one place (the JAR) saves a huge amount of time when deploying larger systems.

Congratulations! You have not only built your first Java file (possibly), but also successfully exported it elsewhere for use. But now that we've built this, how do we use it in the Guvnor?

## Importing the fact model into Guvnor

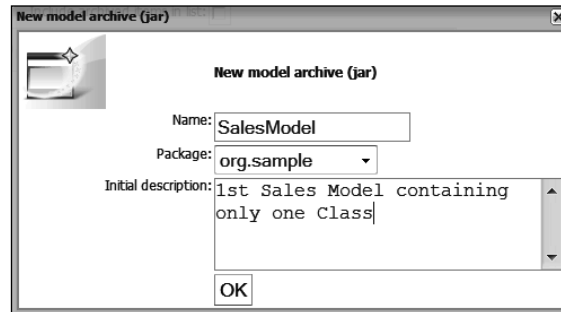
Switching back to the Guvnor, we're now going to create a package to hold our brand new model that we created in the previous steps.



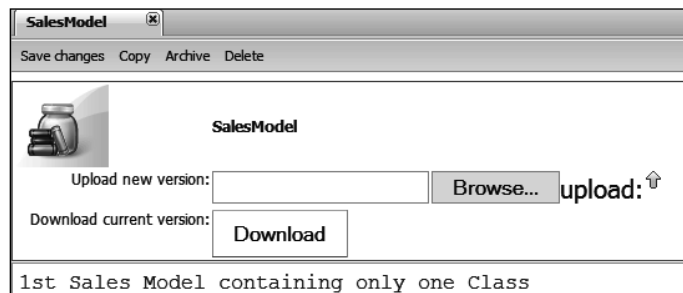
If you want, you can clear out the samples of Hello World from the last chapter. Remember that you can't actually delete the items, but you can archive them. Either way, none of the older samples will get in the way of what we're doing in this chapter.

1. From the lefthand side menu, select the **Packages** tab.
2. Go to **Package | Create New Package (org.sample) | Create Package**.
3. From just below the **Packages** tab, highlight **Create New** and then select the **New model (jar) of fact classes**.

4. Fill out the dialog box that is displayed, as follows. The **Name (SalesModel)** and **Package (org.sample)** should be the same as the ones we created in Eclipse. Click on the **OK** button.



5. Back in the main Guvnor screen, check that everything is in place.

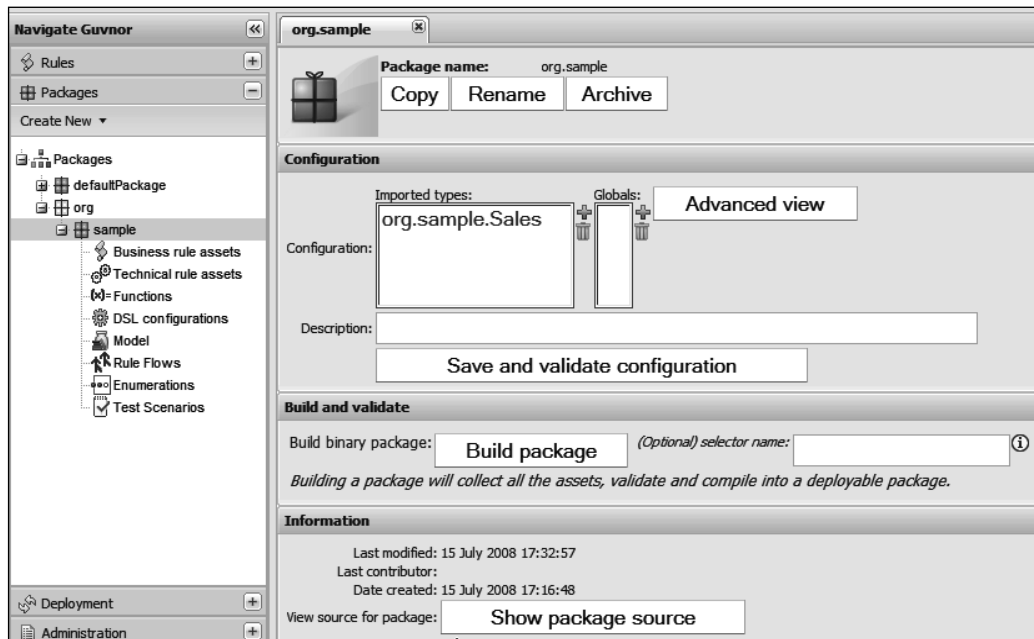


In addition to the **upload** button (that we're going to use in a minute), this screen also has a **Download** button (to retrieve JAR files that you may have uploaded earlier). It also has the usual **Save**, **Copy**, **Archive**, and **Delete** options.

To upload our fact model JAR into Guvnor , follow the steps shown below:

1. Click on the **Browse** button, and then navigate to and select the JAR file that we exported earlier. Click on **OK**.
2. After returning to the screen above, click on the **upload** button (actually, the 'up' arrow icon to the right of **upload**).
3. If everything goes well, you will get the message **File was uploaded successfully**. Click on **OK** to return to the **SalesModel/Package** tab.

4. Save the updated package (by clicking on the **Save Changes** button). As with other saves/check-ins you'll be asked for an optional **checkin** comment.
5. We can check whether Guvnor has successfully picked up the new package information by expanding the **org.package** that was created.



You can see from this example that in the **Configuration** section, under **Imported types**, our class (`org.sample.Sales`) is listed. This means that Guvnor has not only uploaded our class file, but will also allow us to write rules that use this class. Now that Guvnor knows the format of the information that we want to pass in and out, we can start writing rules using Guvnor.

## Guided rules using the fact model

Back at the chocolate factory, we've decided to implement a customer loyalty scheme. When any customer has sales of greater than 100 dollars, we want to give them a flat rate discount of 10 dollars. To put it in a slightly more 'rules-like' format, our new business rule will look something like this:


```
when
    we have a sale greater than 100
then
    Give a discount (by adding a 'negative' sale)
```





Yes, as a business rule it's slightly clunky, but it keeps things simple. In real life we'd just update our sales object with the new balance after the discount (and keep a note of what discount was given).



Of course, we're going to write this rule using the Guvnor. Or rather, you're going to try to write the rule in Guvnor based on the last chapter's tour of the guided editor. I'll show you the full step-by-step answer soon, but the end result will look something similar to the following:

WHEN

Sales 

sales greater than 100  

THEN

Insert Sales  sales -10 

(options)

Some key notes and buttons to use in the Guvnor are:

1. You're going to create a new rule using the guided editor in Guvnor.
2. Click the '+' next to WHEN and THEN to add new conditions/consequences (such as **greater than 100** and **Insert Sales -10**).
3. The 'green arrow' icon allows you to refine these further.
4. Guvnor will pick up the sales model that we imported earlier and offer it as choice to you on a menu.
5. If you make a mistake, the '-' icon allows you to delete a line.

## The step-by-step answer

Before we write a rule we must make sure that we have a category assigned to it. We can use any existing category or we can create a new one (under the **Admin** tab, expand **Categories** | **New Category**). For this step-by-step example, we've created a new **SalesCategory**. But categories are just descriptive tags, so it will work with pretty much any name.

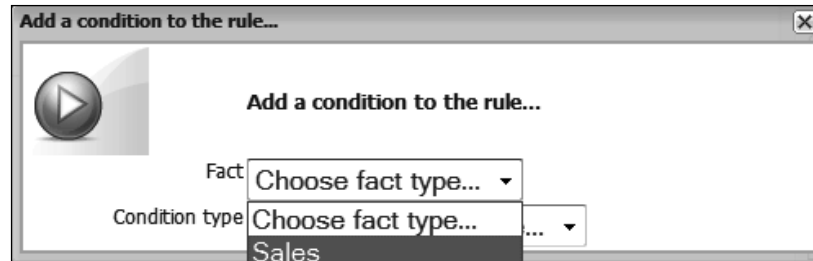
After you've chosen a category, follow these steps:

1. Create a new business rule by selecting menu option **rules | create new | New business rule (guided editor)**.
2. Enter the following values in the screen that is displayed. We will give the new rule a name (**SalesDiscount** – although anything descriptive is OK). We will assign a category (the **SalesCategory** that we created earlier). Then we will pick the package (**org.sample**) from the drop-down list. After entering a description (optional) we need to click on **OK**.

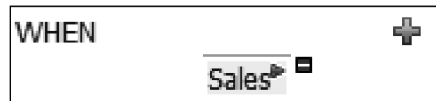
3. We'll then be taken to the guided business rule editor that we saw earlier on our quick tour. In the main section, click on the '+' sign next to the **WHEN** label, to add a condition (that is, to restrict the circumstances under which our rule will fire).

WHEN	+
THEN	+
(options)	+

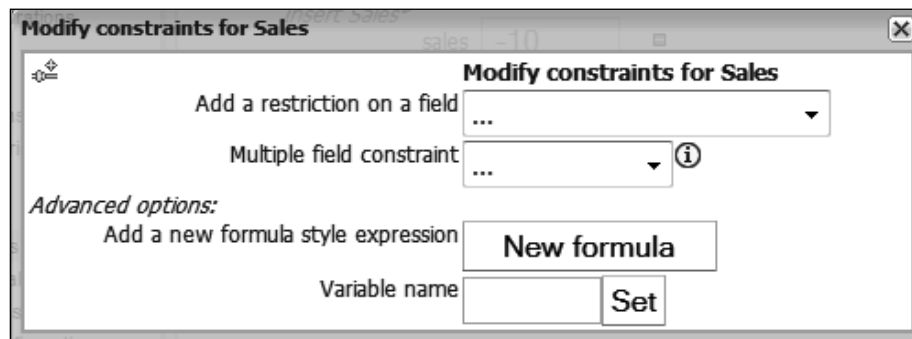
4. The **Add a condition to the rule** dialog will be displayed. We're going to choose a fact type of **Sales**. (Actually, this will be the only fact type in the drop-down list.) This means that our rule will fire only when a sales fact is present. To put it another way, our rule only applies to the rules spreadsheet.



5. After choosing this, we'll automatically be taken back to the guided rule-editing screen.
6. Back on the main screen, we'll see that **Sales** has been added as a condition. Currently, this rule will fire for all sales, so we want to restrict it to only those sales of more than 100 dollars.



7. Because we want to elaborate on the **WHEN** condition, we need to click on the green 'arrow' icon immediately next to the **Sales** condition. We'll be shown the **Modify constraints for Sales** dialog box, shown as follows:



8. The first dropdown field contains a list of all of the fields (columns) available for the **Sales** object. We'll choose **sales** (that is, the dollar value or amount ) from the drop-down list.
9. Back in the guided editor, another line will have been added. The default value is **please choose**. Change this to **greater than or equal to**, as shown in the following diagram.

The screenshot shows a 'WHEN' section in a rule editor. A dropdown menu is open for the 'sales' field, displaying a list of comparison operators: 'is equal to', 'is not equal to', 'is less than', 'greater than', 'less than or equal to', and 'greater than or equal to'. The 'greater than or equal to' option is highlighted. The 'THEN' section is also visible, showing a dropdown menu with 'please choose' as the default value. There are icons for 'Insert', 'options', and 'View source'.

10. Now we need a value to compare this field to (as part of the filter). Click on the pencil icon to set this. In the **Field value** dialog box that is displayed, click on **Literai value**. Literal values are numbers we can enter directly.

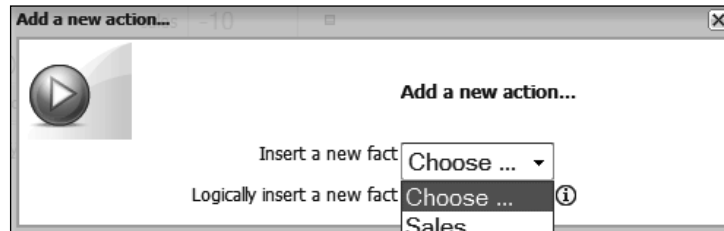
The screenshot shows the 'Field value' dialog box. The 'Literal value' tab is selected, and the text box contains 'Literal value'. The 'Advanced options' section is also visible, showing a text box with 'New formula' entered. There are icons for 'Field value' and 'Advanced options'.

11. Back in the guided editor, a new text box will have appeared. Enter **100** (the value we want to use in our rule) in this text box.

That's it—the WHEN part of the rule is done and should look like the objective picture that we saw at the very start of this sample). Now would be a good time to save the rule (and enter a comment if you see fit).

The THEN part is somewhat easier, in that there are fewer steps to create it.

1. Click on the green plus sign next to the **Then** section. The **Add a new action** dialog box will be displayed:

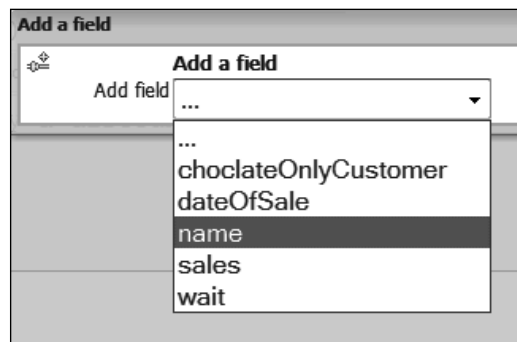


2. We choose to insert a new fact (in the first dropdown). **Sales** will be the only option in this menu. Inserting a new fact is like adding a new row to the Excel Spreadsheet (that is another line of information into the memory).

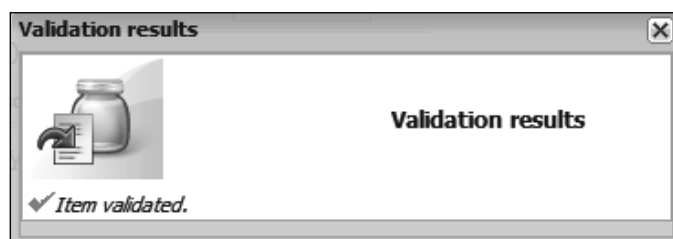


**Logically insert a new fact** does the same thing as inserting a new fact, but automatically removes the row/fact/object as soon as the condition stops being true. For this example, this wouldn't change anything as the **Sales** won't stop being more than 100, but it would make a difference to more dynamic rules (for example, if another rule reduced the price, and we wanted to withdraw the discount in this circumstance).

3. Back in the guided editor, the text **Insert Sales** will now be displayed. Click on the green arrow next to this to begin setting to values for our new sales object (this is similar to saying: once we insert a new line into the Excel Spreadsheet, here are the values that I want to use in the newly created cells).




4. First we're going to add a **name** field. This will appear in the main guided editor. Click on the pencil next to the **name** field to add a value. Enter the value **Discount** (so that the purpose of the new line/new fact is clear).
5. Now that we've created our first field, we can repeat the process for the second one – **Sales**. Add a field (click the green icon next to **Insert sales**) and choose **sales**. Back in the guided editor, click on the pencil next to the new line, and enter a value of **-10**.
6. By now, the screen will look like the screenshot back at the start of this section. Now is a good time to save your rule. Validate the rule by clicking on the **Validate** button. All being well, you will see a dialog box similar to the following example. To close the dialog box, click on the 'x' in the upper-right corner of the dialog box.



Remember that validation is only a check to catch the most obvious errors. It's still possible to get warning messages when we run and test our rule in the next section.

To confirm that the guided editor has written the rule for us, click on the **View source** button. The meaning (give a discount for sales of 100) is pretty much as we'd expect. We've converted the rule into plain English.



Viewing source for: SalesDiscount

```
rule "SalesDiscount"
  dialect "mvel"
  when
    Sales( sales > "100" )
  then
    Sales fact0 = new Sales();
    fact0.setName( " discount" );
    fact0.setSales( -10 );
    insert( fact0 );
  end
```

```
Rule "SalesDiscount"
//name of the rule
// use the slightly more readable mvel
when
    we can find a fact / line of more than 100 sales
then
    create a new line / fact
    set the name to "discount"
    set the sales to -10
    add the line back to the model
end
// end of rule
```

You'll notice that this text is read-only. In the next chapter, we'll show you how to create your rule directly in the text (technical rule) editor. For the moment we've a more pressing problem—how do we try this rule out?

## Running this scenario

The solution is similar to the one we used for running the Hello World example in the previous chapter. The rule we want to exercise is a little bit more complicated, so the scenario that we need to construct is also a little more complicated.

To start, expand the **Package** tab and then create a new test scenario. Give it a name (for example **TestSales**), and select the same package for our rule (**org.sample**). You will be presented (again) with the blank scenario test screen. The scenario that we're going to create is similar to the following screenshot:

The screenshot shows the 'Run scenario' dialog box. It has a 'GIVEN' section with a plus icon, containing an 'insert [Sales]' button, a '[testSales]' label, and input fields for 'sales: 200' and 'name: Acme Corp'. Below this is an 'EXPECT' section with a plus icon, containing a 'Use real date and time' dropdown, an 'Expect rules' section with 'SalesDiscount: fired this many times: 1', a 'More...' button, a '(configuration)' dropdown set to 'All rules may fire', and a '(globals)' section.

The steps for building this screen are similar to those we used before. We use the '+' sign to insert a new **GIVEN/EXPECT**, the small green arrow to refine the scenario, and the '-' sign to remove.

1. Click on the plus sign next to **GIVEN** and choose to insert a new **sales** fact (under any name). Click on **Add**.
2. Click on the **Add a field** button that appears. In the dialog box, select the **name** field.
3. Click on the green arrow next to **sales** to add another field (column in our new row). In this case, use **Sales** and give it a value of **200** (that is, greater than 100!).
4. Click on the green '+' next to **EXPECT**. In the **New Expectation** dialog box that is displayed, click on **show list** and then choose the **Sales Discount** rule.
5. 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.
6. Save this test scenario using the button at the top of the screen.

All being well, if you now click the **Run Scenario** button, you will get a green bar at the top of the screen saying **Results 100%**, along with some additional text: **Rule [SalesDiscount] was activated 1 times**, which indicates that our rule is running as expected.

## What just happened?

The test scenario that we created was equivalent to passing in a spreadsheet with one row (that is, one **Sales Java** object with sales of **200** and a name of **Acme Corp**). We'd expect our sales discount rule to fire under these circumstances and we tell our test scenario to look out for this. When we run the scenario, our rule behaves as expected and fires, giving **Acme Corp** a discount of **-10 Sales** for their order.

## Summary

In this chapter we have covered five main areas. We saw how to get information in and out of our rules, and created the fact model in Java needed to do this. We imported our new fact model into the Guvnor and then built a guided rule around it. Finally, we tested our rule to make sure that it ran correctly.

Using our fact model in a guided rule is a good foundation for the next chapter. In the next chapter, we start writing more powerful text-based rules, starting by using the Guvnor editor, and then moving on to the desktop-based JBoss IDE.



## Where to buy this book

You can buy JBoss Drools Business Rules from the Packt Publishing website:  
<http://www.packtpub.com/jboss-drools-business-rules/book>

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our [shipping policy](#).

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.



[www.PacktPub.com](http://www.PacktPub.com)

For More Information: [www.packtpub.com/jboss-drools-business-rules/book](http://www.packtpub.com/jboss-drools-business-rules/book)