Drools Tutorial

==========

Installation

---

* We will refer to Drools v 7.8.0.Final

Step 1: Download and install **Eclipse IDE for Java Developers**:

https://www.eclipse.org/downloads/packages/

Step 2: Download the folders "**Drools-distribution**" and "**Droolsjbpm-tool-distribution**" and unzip them

https://download.jboss.org/drools/release/7.8.0.Final/

- Open Eclipse.

- Open the menu "**Help**", menu item "**Install new software...**"

- Click on the button "**Add...**" and then "**Local**" to add a new software site.

- Select the folder droolsjbpm-tools-distribution-7.8.0.Final/binaries/**org.drools.updatesite**

- Check the box "**Drools and jBPM**", then click the buttons "Next" and "Finish".

Step 3: Install the Drools runtime:

- Go to "Window/Preferences/Drools/**Installed Drools Runtimes**"

- "Add..." and then "Browse" near the path field

- select the path to ".../drools-distribution-7.8.0.Final/**binaries**"

- Apply and Close

Creating a test project

- Go to “File -> New -> Other…”

- “Drools project”

- Middle option

- Finish

- Right mouse on "DroolsTest" class in source/main/java and “run as -> java application"

- The following should be displayed on the console view:

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

Hello World

Goodbye cruel world

Create a demo

SCENARIO: We are in a bank that handles "bank accounts" and on each account there can be “movements".

The purpose is to calculate the account balance between an "accounting period" of all accounts given the movements it has.

Step1: Create a blank AccountProject of type drools "File -> new -> Other -> Drools project -> Empty project"

The projects will generate "**src/main/java**" and "**src/main/resources**"

Step2: Create 3 java classes inside the package "**com.sample**" of "**src/main/java**":

Account, **AccountingPeriod** and **CashFlow**

* Inside the "rules" package of "src/main/resources" we will create the .drl files containing the rules

Step3: In the Account class, add two "private" attributes:

1. **accountNumber** (long) and **balance** (double)
2. autogenerate generate **getters**/**setters** (using Source/...)
3. autogenerate the **toString**() method

Step4: In the AccountingPeriod class add "private" attributes:

1. **startDate** (java.util.Date) and **endDate** (Date)
2. autogenerate **getters**/**setters** (using Source/...)
3. autogenerate the **toString**() method

Step5: In CashFlow add "private" attributes:

1. **mvtDate** (java.util.Date), **amount** (double), **type** (int) and **accountNumber** (long)
2. autogenerate **getters**/**setters** (using Source/...)
3. autogenerate the **toString**() method

Test the first functionalities and rules

You download the entire project at https://github.com/AlessandroMarcellettiUnicam1/Drools_lecture

* We are going to add a new drl file (drl = drools rule language)

Step0: In the package "**rules**" in "src/main/resources" create a rule file called "**lesson1.drl**"

using “New -> Other -> Drools -> Rule Resource” specifying the name and the package (com.sample)

Step1: Adding a simple condition to a rule

1. in the previously created rule, import the Account classes:

```
import com.sample.Account
import com.sample.CashFlow
import com.sample.AccountingPeriod
```

2. Modify the first rule with a condition that is just a fact of type Account. If the rule is fired, then we shall show the message "The account exists" in the console:

```
rule "Your First Rule"
    when
       Account(  )
         //conditions
    then
        System.out.println("rule1: The account exists");
End
```

3. Create a class **AccountTest** in src/main/java **com.sample** package to test this:

```
package com.sample;

import java.text.SimpleDateFormat;
import org.kie.api.KieServices;
import org.kie.api.event.rule.ObjectDeletedEvent;
import org.kie.api.event.rule.ObjectInsertedEvent;
import org.kie.api.event.rule.ObjectUpdatedEvent;
import org.kie.api.event.rule.RuleRuntimeEventListener;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.rule.FactHandle;

public class AccountTest {

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

            // Step1 - go !
```

```
            Account a = new Account();
            kSession.insert(a);
            kSession.fireAllRules();

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

4. Executing it we obtain in the console: The account exists

NB: drl files represent the "production memory", while the KieSession the "working memory" where we can add facts to it with the method "insert".

Step2: Using callback to log activity in drools runtime

Create a function createEventListener containing the listener implementation for logging

```
public static void createEventListener(KieSession kSession) {
        RuleRuntimeEventListener listener = new RuleRuntimeEventListener() {

            // Listen to the update event
            public void objectUpdated(ObjectUpdatedEvent arg0) {
                System.out.println("*****Object Updated*****\n"
                        +arg0.getObject().toString());

            }

            // Listen to the insert event
            public void objectInserted(ObjectInsertedEvent arg0) {
                System.out.println("*****Object inserted***** \n"
                        + arg0.getObject().toString());

            }

            // Listen to the delete event
            public void objectDeleted(ObjectDeletedEvent arg0) {
                System.out.println("*****Object Retracted*****\n"
                        + arg0.getOldObject().toString());

            }
        };

        // Add the event listener to the session
        kSession.addEventListener(listener);
    }
```

* This listener will log an event every time a fact is **inserted**, **updated** or **retracted** showing it on the console. Here we are using the toString method of the java instance given ObjectActionEvent.getObject().toString().

Step3: test all event types

For triggering all the different event types we need to create, update and delete some information. We can do this by creating a new function `testAllEvents` containing these operations

```
public static void testAllEvents(KieSession kSession) {
               Account a = new Account();
               // 1 - create an account instance with the accountNumber
               a.setAccountNumber(10);
                FactHandle handlea = kSession.insert(a);
                // 2  - update the balance value of the account instance
                a.setBalance(12.0);
                kSession.update(handlea, a);
                // 3 - delete the account instance
                kSession.delete(handlea);
                // Evaluate rules
                kSession.fireAllRules();
                System.out.println("\nClose...");

                /*
                     Account a = new Account();
                       kSession.insert(a);
                       kSession.fireAllRules();
                       kSession.fireAllRules();

                     The rule is fired only once.

                       Account a = new Account();
                       kSession.insert(a);
                       kSession.fireAllRules();
                       a.setAccountNo(10);
                       kSession.fireAllRules();

                     The rule is fired only once.

                       Account a = new Account();
                       a.setAccountNo(10);
                       FactHandle handlea = kSession.insert(a);
                       kSession.fireAllRules();
                       a.setBalance(12.0);
                       kSession.update(handlea, a);
                       kSession.fireAllRules();

                     The rule is fired twice.
                 */
}
```

When you want to update an object, you first must memorize the fact handle, the same applies when you want to retract.

This is the **output**:

```
*****Object inserted*****
Account [accountNumber=10, balance=0.0]
*****Object Updated*****
Account [accountNumber=10, balance=12.0]
*****Object Retracted*****
Account [accountNumber=10, balance=12.0]
```

```
Close...
```

As you can notice the rule is not fired because the fact Account was deleted.

Here is what is happening when the FireAllRules method is called on a stateful session :

1. drools will look at all rules that can apply and put it in its agenda
2. drools will execute the rule that is on top of its agenda
3. once fired, the rule will be deactivated

This means that we have to inform drools about a state change in one of the facts in the **when** part (lhs) to make him reconsider the rule

Step4: Use the rules actions for inserting/updating/deleting facts.

1. inside the **lesson1.drl** add 2 new rules

```
rule "Your Second Rule"

    when
        CashFlow(  )
    then
        System.out.println("cash flow exists, inserting an account period");
        AccountingPeriod newPeriod = new AccountingPeriod();
        insert (newPeriod);
end

rule "Your Third Rule"
    when
        AccountingPeriod(  )
    then
        System.out.println("rule3: Accounting period exists");
End
```

2. Create a function `testCashFlow`

```
public static void testCashFlow(KieSession kSession) {
         CashFlow a = new CashFlow();
         FactHandle handlea = kSession.insert(a);
         kSession.fireAllRules();
         System.out.println("Close...");
}
```

Output:

```
*****Object inserted*****
CashFlow [mvtDate=null, amount=0.0, type=0, accountNumber=0]
rule2: The cash flow exists, inserting an account period
*****Object inserted*****
AccountingPeriod [startDate=null, endDate=null]
rule3: Accounting period exists

Close...
```

Test Bindings

Binding facts in the rule condition to the rule actions.

We want to update the account balance for each **CashFlow**.

We first put the CashFlow and select all CashFlow of type **CREDIT**.

To do so, we add a constraint on the "**type**" attribute of java class CashFlow.

Step1: Add constants variables to CashFlow:

```
public static int CREDIT = 1;
public static int DEBIT = 2;
```

* Comment all rules in lesson1.drl

Step2: Create a new file lesson2.drl with the following rule:

```
rule "Credit rule"

    when
        $cash : CashFlow(type == CashFlow.CREDIT )
        $acc : Account(  )
    then
       // If the cashflow is credit update the balance of the account by adding
       // the amount of the cashflow
        $acc.setBalance($acc.getBalance() + $cash.getAmount());
        System.out.println("Account no " + $acc.getAccountNumber()+
        "has now a balance of " + $acc.getBalance());
end
```

Step3: Create a function testBindings:

```
public static void testBindings(KieSession kSession) throws Exception {
         // 1 - create an account instance with some default values
         Account a = new Account();
        a.setAccountNumber(1);
        a.setBalance(0);
        kSession.insert(a);

        // 2 - create a cashFlow instance with some default values
        CashFlow cash1 = new CashFlow();
        cash1.setAccountNumber(1);
        cash1.setAmount(1000);
        cash1.setType(CashFlow.CREDIT);

        // 3 - create a second instance with the credit type
        CashFlow cash2 = new CashFlow();
```

```
        cash2.setAccountNumber(2);
        cash2.setAmount(500);
        cash2.setType(CashFlow.CREDIT);

        // 4 - create a cashflow instance with the debit type
        CashFlow cash3 = new CashFlow();
        cash3.setAccountNumber(1);
        cash3.setAmount(500);
        cash3.setType(CashFlow.DEBIT);

        kSession.insert(cash1);
        kSession.insert(cash2);
        kSession.insert(cash3);

        kSession.fireAllRules();
        System.out.println("Balance of account n. "
        + a.getAccountNumber() + ": " + a.getBalance()); //1000,0
        System.out.println("Close...");

}
```

Step4: Add a new rule to trigger the debit rule:

```
rule "Debit rule"

    when
        $cash :CashFlow(type == CashFlow.DEBIT )
        $acc : Account(  )
    then
        $acc.setBalance($acc.getBalance()- $cash.getAmount());
       System.out.println("Account no "+ $acc.getAccountNumber() +" has now a balance
      of   "+ $acc.getBalance());
End
```

PROBLEM: cash2 is applied to accountNumber 1, despite "cash2.setAccountNumber(2)"

We will resolve this with **Attribute binding.**

We want to link facts each other using their attributes.

Step5: Add rules for using the data binding

```
rule "Credit rule with binding"

    when
        $cash : CashFlow( $no : accountNumber, type == CashFlow.CREDIT )
        $acc : Account(accountNumber == $no  )
    then
             $acc.setBalance($acc.getBalance() + $cash.getAmount());
        System.out.println("Account no " + $acc.getAccountNumber() + " has now a
balance of "+$acc.getBalance());
     end
```

Step6: Add rules for date evaluation

```
rule "Credit rule"

    when
        $cash : CashFlow( $aDate : mvtDate, $no : accountNumber, type ==
CashFlow.CREDIT )
        $acc : Account(accountNumber == $no  )
        $period : AccountingPeriod(startDate <= $aDate && endDate >= $aDate)
    then
             $acc.setBalance($acc.getBalance() + $cash.getAmount());
        System.out.println("Account no "+$acc.getAccountNumber() + " has now a balance
of "+$acc.getBalance());

end

rule "Debit rule"

    when
        $cash : CashFlow( $aDate : mvtDate, $no : accountNumber, type == CashFlow.DEBIT
)
        $acc : Account(accountNumber == $no  )
        $period : AccountingPeriod( startDate <= $aDate && endDate >= $aDate)
    then
        $acc.setBalance($acc.getBalance() - $cash.getAmount());
        System.out.println("Account no " + $acc.getAccountNumber() + " has now a
balance of " + $acc.getBalance());

End
```

We also add a constraint so that the **mvtDate** of the **CashFlow** is between the **startDate** and **endDate** of the **AccountingPeriod**.

Step7: Create a function setDates for setting the dates and for creating an AccountingPeriod

```
public static void setDates(CashFlow cash1, CashFlow cash2, CashFlow cash3, KieSession
kSession) throws Exception {
             cash1.setMvtDate(new SimpleDateFormat("yyyy-MM-dd").parse("2021-01-1"));
             cash2.setMvtDate(new SimpleDateFormat("yyyy-MM-dd").parse("2021-02-1"));
             cash3.setMvtDate(new SimpleDateFormat("yyyy-MM-dd").parse("2021-05-1"));

             AccountingPeriod period = new AccountingPeriod();
        period.setStartDate(new SimpleDateFormat("yyyy-MM-dd").parse("2021-01-01"));
        period.setEndDate(new SimpleDateFormat("yyyy-MM-dd").parse("2021-04-1"));
        kSession.insert(period);
}
```

OTHER TYPES OF RULES

```
//created on: 8 mar 2021
package com.sample

//list any import classes here.
import com.sample.Account
import com.sample.CashFlow
import com.sample.AccountingPeriod


//declare any global variables here



// * "in" Constraint

rule "The cashFlow can be a  credit or a debit"

    when
        $cash :CashFlow(type in ( CashFlow.DEBIT,CashFlow.CREDIT) )

    then
        System.out.println("The cashFlow is a credit or a debit");
end



// * Nested Accessor This allows to add a constraint to an attribute class without the
need to add the linked object to the session.

rule "Accessor"
    when
        $cash :PrivateAccount( owner.name =="Héron" )
    then
        System.out.println("Account is owned by Héron");
end



// * And/or It is possible to do constraints on attribute like in java.

rule "infixAnd"
    when
      ( $c1 : Customer ( country=="GB") and  PrivateAccount(  owner==$c1))
            or
       ( $c1 : Customer (country=="US") and PrivateAccount(  owner==$c1))
    then
        System.out.println("Person lives in GB or US");
end


// * not This allows to test if no fact of a type is in the session.

rule "no customer"
    when
        not Customer(  )
    then
```

```
        System.out.println("No customer");
end


// * exist On the contrary of previous syntax, this allows to test if there at least
one fact type is in the session.

rule "Exists"
    when
        exists Account(  )
    then
        showResult.showText("Account exists");
end



// * ForAll We would like to verify that every cashflow instance is linked to an
Account instance.

rule "ForAll"
    when
        forall (   Account( $no : accountNo  )
                    CashFlow( accountNo  == $no)
                   )
    then
        showResult.showText("All cashflows are related to an Account ");
end
```