Labs
« Drools »

Prerequisites:

- Good internet connection
- JDK11
- Git
- Eclipse
- Maven
- Optional Docker

Github solutions
https://github.com/dthibau/drools-solutions

Lab 1 : Set up the IDE

1.1 Eclipse Plugin and project creation wizard

See the tutorial
https://wwu-pi.github.io/tutorials/lectures/lsp/030_install_drools.html

Check that you can create a Drools project via the Eclipse wizard.
Run the examples specifically:
The decision table
The DRL file

Display views and editors associated with Drools
Run at the debugger and inspect the kSession object

1.2 Archertype Maven

With the command line:

```
mvn archetype:generate -DgroupId=org.formation -DartifactId=tp0-maven -
DarchetypeArtifactId=kie-drools-archetype -DarchetypeVersion=7.73.0.Final -
DarchetypeGroupId=org.kie -DinteractiveMode=false
```

Then execute ***mvn package***

Import the maven project in Eclipse

Optional: Eclipse m2Eclipse Maven plugin doesn't know how to handle kie Maven plugin; this leads to a red cross or warnings
Possibly to make this cross disappear add the following lines in the *pom.xml*

```
            <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.eclipse.m2e</groupId>
                    <artifactId>lifecycle-mapping</artifactId>
                    <version>1.0.0</version>
                    <configuration>
                        <lifecycleMappingMetadata>
                            <pluginExecutions>
                                <pluginExecution>
                                    <pluginExecutionFilter>
                                        <groupId>org.kie</groupId>
                                        <artifactId>kie-maven-
plugin</artifactId>
                                        <version>${drools-version}</version>
                                        <goals>
                                            <goal>build</goal>
                                        </goals>
                                    </pluginExecutionFilter>
                                    <action>
                                        <ignore />
                                    </action>
                                </pluginExecution>
                            </pluginExecutions>
                        </lifecycleMappingMetadata>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
```

View and run the test class

Lab 2 : Stateless and stateful sessions

2.1 Stateless Session

Import the provided Maven project

Implementing an Helper class

The *org.formation.helper.RuleRunner class* is a utility that should allow to start a stateless or stateful session.

Complete the provided source file

Interaction with a stateless session

The problem is to write a rules file to calculate the price of car insurance.
The base price is calculated as follows

- If the driver is a young driver (between 18 and 25 years old), the basic price is 500€
- If the driver is a confirmed driver (> 25 years old), the basic price is 300€

A discount is made based on the seniority of the customer:

- If the client has 5 years of seniority, he is granted a 10% discount
- If the customer has 10 years of seniority, he is granted a 20% discount

Finally, a penalty of 5% is applied per number of incidents the driver has had.

Implement the rule file and test it with the class *org.formation.drools.test.AssuranceTest*

2.2 Stateful Session

This part takes the example of the course material
The rules to implement are:

- As soon as a fire breaks out in a room, an alarm must be triggered. If a sprinkler exists for this room, the sprinkler must activate.
- If there is no fire in any of the rooms, a message indicating that all is well must be displayed

Work to do :

RuleRunner class
In the RuleRunner class create a stateful session associated with:

- A KieRuntimeLogger generating a stateful.log trace file
- DebugRuleRuntimeEventListener and DebugAgendaEventListener debug listeners
- A ConsoleChannel channel (to be implemented) which displays an object on the standard console

Complete this method `public FactHandle[] insertFacts(Object[] facts){`

DRL file

Write the previously described rules in a DRL

Test class
Comlete and use ***org.formation.test.TestFire*** to validate your rules

Lab 3 : Conflicts, ordering rules, type declaration, timers...

Problem

This lab resumes the stateful part of the previous lab

We want to complete the previous lab by adding the following rules

- The triggering of the alarm must be delayed by 5s in case the fire alert is a false alarm
- When 2 sprinklers are activated, we would like to go into panic mode and activate all available sprinklers
- Panic mode should display a general evacuation message
- We would like to display a message when all Sprinklers are activated
- As long as the alarm is in progress, we would like to periodically send an SMS to the building manager

Rule's organization

Rules are partitioned into several groups:

- The rules in ***normal*** mode: you activate as many Sprinklers as fires
- The rules in ***panic*** mode : we force the activation of all the Sprinklers
- The rules displaying the status of the building (Everything is fine or all the Sprinklers are on)
- The rules managing the alarm (Recurrent alarm triggering and notification)

The first two groups are exclusive, only 1 type of rule applies depending on the mode.
They will therefore be implemented with the attribute ***agenda-group***

Rules based on the number of fires will determine the mode. They must be applied first
The mode could be a type declared type in the DRL file:

```
declare PanicMode
 on : boolean
end
```

A rule gurantees that there is always a singletong fact of PanicMod in the working memory:

```
rule "Initialiser le mode panic à false"
      when
            not PanicMode()
      then
            insert (new PanicMode());
            System.out.println( "On initialise l'immeuble"  );
end
```

2 other rules will determine the mode of the building based on the number of lights present, these rules will be responsible for activating the appropriate agenda group.

Rules will be triggered by *fireUntilHalt()* to use the timers.

The new class *RuleRunner and* test classes are provided.

Lab4 : Decision tables and templates

4.1 Decision table

We take again the *stateless* part of the TP1 and we implement it via a decision table

Use the wizard to start with a pre-filled spreadsheet

4.2 Templates

In the 2nd part of the workshop we will generate a DRL file from a template and a data collection (stored in a Map)

Retrieve the provided sources, understand it and complete the template provided

Lab5 : DMN

Pre-requesite : VSCode and [Red Hat provided DMN Editor extension]

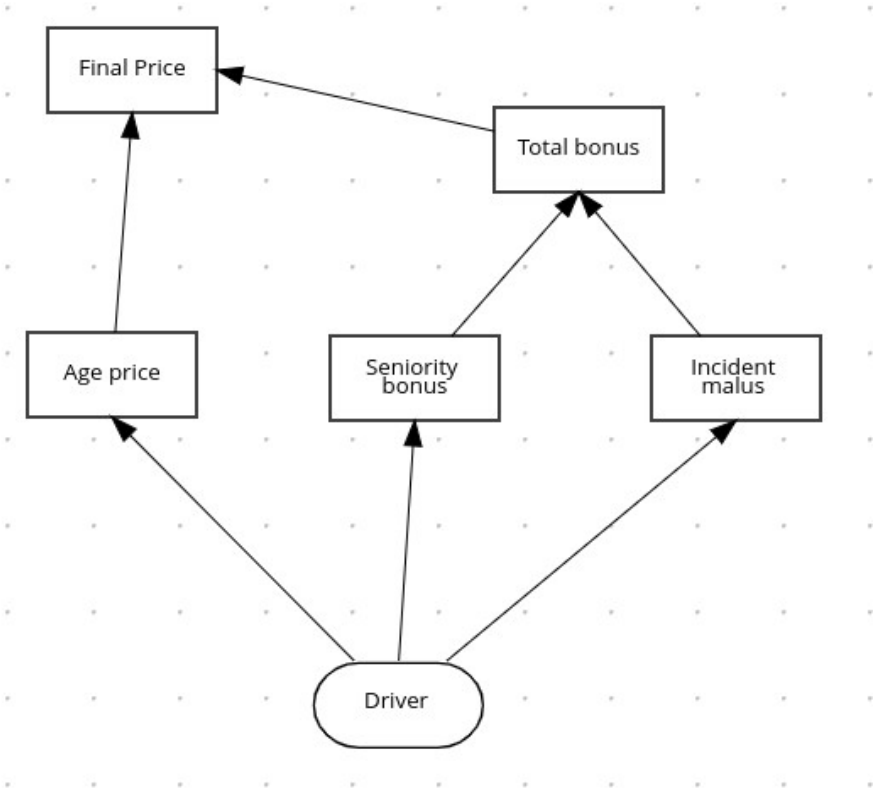
Create a project :

```
mvn archetype:generate -B -DarchetypeGroupId=org.kie -
DarchetypeArtifactId=kie-drools-archetype -
DarchetypeVersion=7.73.0.Final -DgroupId=org.formation -
DartifactId=dmn-project -Dversion=1.0-SNAPSHOT -
Dpackage=org.formation
```

Delete the sample files and create the folder ***dmn*** under ***src/main/resources***

Create a file named insurance.dmn and edit it with VSCode

Create the DMN diagram implementing the stateless rules of Lab2



Add the following dependencies in your Maven project

```
<dependencies>
  <dependency>
    <groupId>org.kie</groupId>
```

```
    <artifactId>kie-dmn-core</artifactId>
    <version>${drools-version}</version>
  </dependency>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-ci</artifactId>
    <version>${drools-version}</version>
  </dependency>
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-compiler</artifactId>
    <version>${drools-version}</version>
  </dependency>
</dependencies>
```

Then create a test class which validates the rules.

Here is a sample of the code needed :

```
KieServices ks = KieServices.Factory.get();
KieContainer kContainer = ks.getKieClasspathContainer();
DMNRuntime dmnRuntime =
KieRuntimeFactory.of(kContainer.getKieBase()).get(DMNRuntime.class);
String namespace = "https://kiegroup.org/dmn/<<UUID>>";
// replace the value of namespace; copy it form your DMN model
String modelName = "age-classifier";
DMNModel dmnModel = dmnRuntime.getModel(namespace, modelName);
DMNContext dmnContext = dmnRuntime.newContext();
for (Integer age : Arrays.asList(1,12,13,64,65,66)) {
    dmnContext.set("age", age);
    DMNResult dmnResult = dmnRuntime.evaluateAll(dmnModel, dmnContext);
    for (DMNDecisionResult dr : dmnResult.getDecisionResults()) {
      System.out.println("Age: " + age + ", " +
                "Decision: '" + dr.getDecisionName() + "', " +
                 "Result: " + dr.getResult());
           }
       }
```

Lab6 : CEP

Import the provided Maven project

Inpect the code

Write a rule that detects an HeartAttack:
If there is no heartbeat within a sliding window of 5 seconds

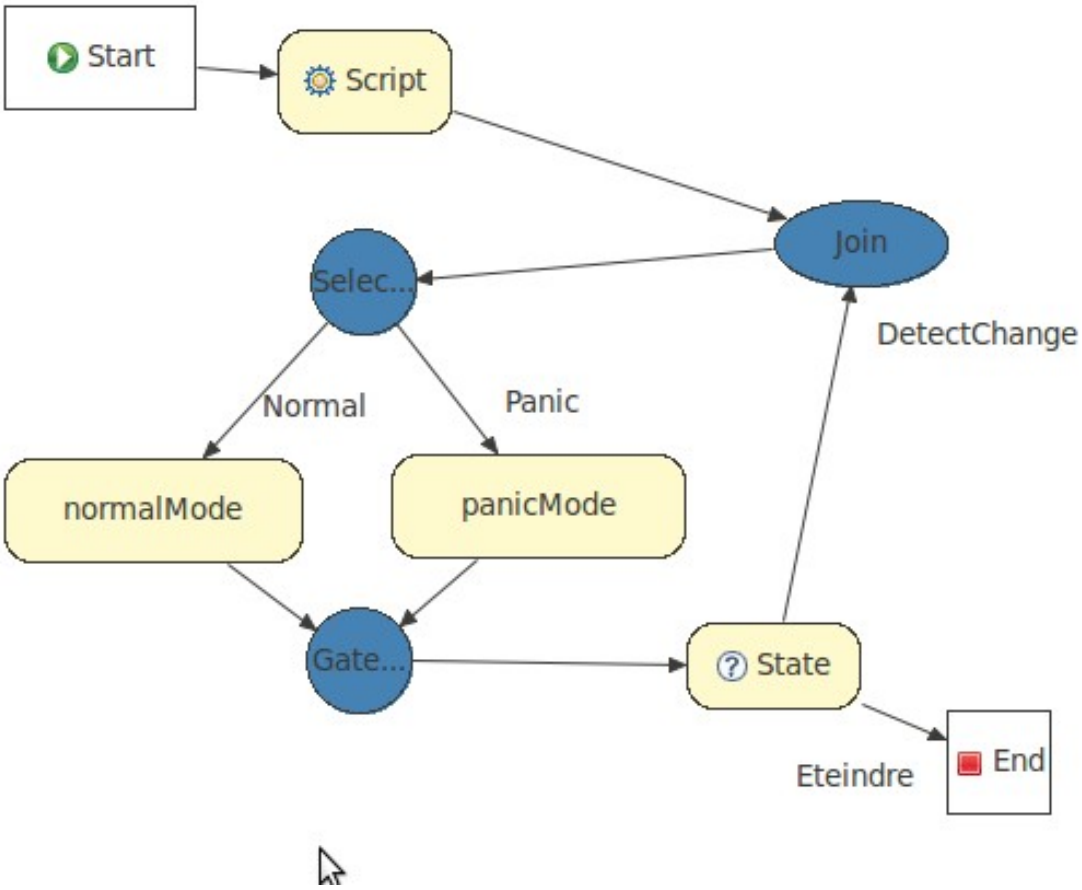
Run the tests, they should detect an event ***HeartAttackEvent***

Lab7 : Drools and processes

In this exercise, we will reproduce the Lab2 using a jbpm process and rule task.

Problematic

The building's alarm system has two modes: the normal mode and the panic mode. Instead of implementing this behavior with group calendars we implement it via the following process:



The alarm system when it is started performs an Initialization task (Script) and then according to the facts present in the working memory is positioned on one of the nodes of type RuleTask (***normalMode*** or ***panicMode***).
The entry into one of these nodes triggers the application of the corresponding rules, then the process goes into a pending state (*signal catch*).
When receiving a DetectChange event, the process loops and redetermines the alarm mode
When the Shut Down event is received, the alarm system is disconnected.

Implement the ***fire.bpmn*** process and the ***fireMode.drl*** rules to get the desired behavior.
The other necessary classes are provided.

Lab8 : Maven Knowledge repository and KieScanner

In this lab, we reorganize the previous projects dents.

Sources refactoring

We reorganize the sources of a Drools project in 2 Maven project:

- ***8_Rules*** : Contains the domain model classes and the rules. This project is dependent on drools-compiler and drools-decisiontables
- ***8_Application*** : Contains a main class that scans the Maven repository every 10s to retrieve the *kjar* from the first project. This project is dependent on kie-api, drools-compiler, drools-decisiontables and kie-ci

Build

Build the 8_rules project and deploy it in the local Maven repository
mvn install

In *8_application* execute the rules

Update

Make a change in the rules, deploy and observe the dynamic loading of the new rules

Lab9 : KieServer and Workbench

7.1. Installation KieServer

Option Wildfly
Download Wildfly 14 and unzip

Download a distribution of KieServer and place it in:
WILDFLY_HOME/standalone/deployments

Add user with ***kie-server*** role:
WILDFLY_HOME/bin/add-user.[sh|bat] -a -u 'kieserver' -p 'kieserver1!' -ro 'kie-server'

Start the server in full configuration
WILDFLY_HOME/bin/standalone.[sh|bat] -c standalone-full.xml

Check the URL :
http://SERVER:PORT/CONTEXT/services/rest/server/

Go to swagger docs and try some REST calls

Option Docker
Use demo image:

```
docker run -p 8080:8080 quay.io/kiegroup/kie-server-showcase:latest
```

Browse
http://localhost:8080/kie-server/services/rest/server/

The user is **kieserver /kieserver1!**

To deploy a knowledge base,

- Create an empty directory***myM2***
- Restart a new container by mounting this directory to ***opt/jboss/.m2***
  -v myM2 : opt/jboss/.m2
- Create a settings.xml (Maven) so that your local repository points to ***myM2***
  ```
  <settings>
  <localRepository><myM2>/repository</localRepository>
  </settings>
  ```
- Deploy the knowledge base of Lab8
  mvn -s settings.xml install

7.2. Deployment of the rule project via the workbench

7.2.1 Installation of the workbench

Download the workbench and deploy it by placing it in:
WILDFLY_HOME/standalone/deployments

Or Docker option:

```
docker run -p 8080:8080 -p 8001:8001 -d \
   -v /home/myuser/wb_git:/opt/jboss/wildfly/bin/.niogit:Z \
   --name drools-workbench jboss/drools-workbench-showcase:latest
```

Try the samples:
https://docs.drools.org/7.73.0.Final/drools-docs/html_single/index.html#_wb.quickstart

2.1 Importing assets
Add a new project named "Assurance -DRL" and import the following assets paying close attention to the package names:

- *Assurance.drl*
- The 2 model classes

Build and deploy to KIE server

7.3. Project client

In the Eclipse IDE, create a project that invokes the Drools engine and the *"Assurance -DRL"* container

Lab Annex : DSL

Rule file

We complete the *stateless* part of Lab1.
We wan to express the rules in natural langage :

```
règle "Prix de base Jeune conducteur"
      si
            L age du conducteur est compris entre 18 et 25
            Pour une assurance
      alors
            Le prix de base est de 500.00
fin
```

The complete rules file is provided

Implementing the DSL file

After creating the Drools Lab3 project, use the wizard to create the DSL file, put it in the same directory as the rule file whose extension has been changed to .dslr

Write the necessary associations

Editing the .dslr file

Add the ***expander*** statement to the top of the file that references the DSL file.

Rewrite the rules using the "business" syntax

The editor allows to have the business view and the technical view

Run the test class

Lab Annex : Performance

4.1 Rete network

Prerequisites: Graphviz installation https://graphviz.org/

Get the sources provided.

The PhreakInspector class generates a graph in graphviz format

Run it on your drl files.

Visualize the graph with graphviz:
dot -Tsvg /tmp/phreak.dot > /tmp/phreak.svg

4.2 Drools Metric

Reference tutorial:
https://blog.kie.org/2021/07/how-to-find-a-bottle-neck-in-your-rules-for-performance-analysis.html

Import the provided project and view:

- Dependence on drools-metric
- The JoinTest test class

Run the test class and observe the console:

- ***dumpRete output*** : Represents the Rete network
- ***dumpAssociatedRulesRete output*** : Allows you to correlate segments to rules

Then activate the provided logback file by removing the .bak extension, run the test

You notice that
[AccumulateNode(8)], evalCount:4950000, elapsedMicro:1277578
is anormal
The output of dumpAssociatedRules(). Show that AccumulateNode(8)] is associated with a rule [Combinaison de collecte des commandes coûteuses]
=> We must review this rule

If evalCount is abnormally large, look at the previous node's log:
[JoinNode(7) - [ClassObjectType class=com.sample.Order]], evalCount:100000, elapsedMicro:205274.
Accumulate is evaluated for each $o1/$o2 combination produced by the JoinNode. Which is not good.
This accumulation is only used to calculate the maximum price of a customer's orders.
It should therefore only be assessed once per client.

Correct this defect, rerun the test and identify the next bottleneck

Make a correction to optimize the overall time