Cahier de TP
« Drools 7»

Pré-requis :

- Bonne connexion Internet
- JDK11
- Git
- IDE Java
- Maven

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

Atelier 1 : Mise en place IDE

1.1 Plugin Eclipse et assistant de création de projet

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

Vérifier que vous pouvez créer un projet Drools via l'assistant Eclipse.
Exécuter les exemples en particulier :
La table de décisions
Le fichier DRL

Afficher les vues et éditeurs associés à Drools
Exécuter au debugger et inspecter l'objet kSession

1.2 Création de projet Maven

Via la commande en ligne :

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

Effectuer ensuite un ***mvn package***

Importer le projet Maven dans Eclipse

Optionnel : le plugin Maven d'Eclipse m2Eclipse ne sait pas comment gérer le plugin de kie Maven plugin ; cela provoque une croix rouge ou des warnings
Éventuellement pour faire disparaître cette croix ajouter les lignes suivantes dans le 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>
```

Visualiser et exécuter la classe de test

Atelier 2 : Prise en main de l'API

2.1 Session stateless

Création du projet et installation des sources

Importer le projet Maven fourni

Mise au point de la classe Helper

La classe *org.formation.helper.RuleRunner* est un utilitaire qui doit permettre de démarrer une session *stateless* ou *stateful.*

Compléter le fichier source fourni

Interactions avec une session stateless

La problématique consiste à écrire un fichier de règles permettant de calculer le prix d'une assurance auto.
Le prix de base est calculé comme suit :

- Si le conducteur est un jeune conducteur (entre 18 et 25 ans), le prix de base est de 500€
- Si le conducteur est un conducteur confirmé (> 25ans), le prix de base est de 300€

Une remise est effectuée en fonction de l'ancienneté du client :

- Si le client a 5 ans d'ancienneté, on lui accorde une remise de 10%
- Si le client a 10 ans d'ancienneté, on lui accorde une remise de 20%

Enfin, un malus de 5% est appliqué par nombre d'incidents qu'a eu le conducteur.

Implémenter le fichier de règle et le tester avec la classe *org.formation.drools.test.AssuranceTest*

2.2 Session stateful

Cette partie reprend l'exemple du support de cours
Les règles à implémenter sont les suivantes :

- Dés qu'un feu se déclare dans une pièce, une alarme doit être déclenchée. Si un sprinkler existe pour cette pièce, le sprinkler doit s'activer.
- Si il n'y a de feu dans aucune des pièces, un message indiquant que tout va bien doit s'afficher

Travail à faire :

Classe *RuleRunner*
Dans la classe RuleRunner créer une session stateful associé à :

- Une *KieRuntimeLogger* générant un fichier de trace *stateful.log*

- Les listeners de debug *DebugRuleRuntimeEventListener* et *DebugAgendaEventListener*
- Un channel *ConsoleChannel* (à implémenter) qui affiche un objet sur la console standard

Compléter la méthode `public FactHandle[] insertFacts(Object[] facts){`

Fichier DRL
Écrire les règles précédemment décrites dans un DRL

Classe de test
Compléter et utiliser la classe de test ***org.formation.test.TestFire*** pour valider vos règles

Atelier 3 : Conflits, séquencement de règles, déclaration de type, timers, ...

Problématique

Ce TP reprend la partie *stateful* du TP précédent

Nous voulons compléter le TP précédent en ajoutant les règles suivantes

- Le déclenchement de l'alarme doit être différé de 5s au cas ou l'alerte au feu serait une fausse alerte
- Lorsque 2 sprinklers sont activés, nous voudrions passer en mode ***panique*** et activer tous les sprinkler disponibles
- Le mode *panique* doit afficher un message d'évacuation générale
- Nous voudrions afficher un message lorsque tous les Sprinkler sont activés
- Tant que l'alarme est en cours, nous voudrions périodiquement envoyer un SMS au responsable de l'immeuble

Éléments de solution

Typiquement, les règles sont partitionnées en plusieurs groupes :

- Les règles en mode *normal* : on allume autant de Sprinkler que de feux
- Les règles en mode *panique* : on force l'activation de tous les Sprinkler
- Les règles affichant le statut de l'immeuble (Tout va bien ou tous les Sprinkler sont enclenchés)
- Les règles gérant l'alarme (Déclenchement de l'alarme et notification récurrentes)

Les deux premiers groupes sont exclusifs, 1 seule types de règles s'applique en fonction du mode. Ils seront donc implémentés avec l'attribut ***agenda-group***

Des règles s'appuyant sur le nombre de feu devant s'appliquer le plus tôt détermineront le mode. Le mode pourra être un type déclaré directement dans le fichier de règles :

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

On s'assurera qu'un fait de type *PanicMode* sera toujours présent dans la mémoire de travail :

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

2 autres règles détermineront le mode de l'immeuble en s'appuyant sur le nombre de feux présents, ces règles seront responsable d'activer le groupe d'agenda adéquat.

Les règles sur les sessions seront déclenchées par *fireUntilHalt()* afin de déclencher les timers. La nouvelle classe *RuleRunner* ainsi que la classe de test sont fournies.

Ateliers 4 : Performance

4.1 Réseau Rete

Pré-requis : Installation graphviz *https://graphviz.org/*

Récupérer les sources fournis.

La classe ***PhreakInspector*** permet de générer un graphique au format *graphviz*

Exécuter le sur vos fichiers drl.

Visualiser le graphique avec graphviz :
dot -Tsvg /tmp/phreak.dot > /tmp/phreak.svg

4.2 Drools Metric

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

Importer le projet fourni et visualiser :

- La dépendance sur Drools-metric
- La classe de test *JoinTest*

Exécuter la classe de test et observer la console :

- ***dumpRete output*** : Représente le réseau Rete
- ***dumpAssociatedRulesRete output*** : Permet de corréler les segments aux règles

Activer ensuite le fichier logback fourni en enlevant l'extension .bak, exécuter le test

Vous constatez que
[AccumulateNode(8)], evalCount:4950000, elapsedMicro:1277578
est anormal.
La sortie de dumpAssociatedRules(). Montre que AccumulateNode(8)] est associé à une règle [Combinaison de collecte des commandes coûteuses]
=> Il faut revoir cette règle.

Si evalCount est anormalement grand, il faut regarder le journal du nœud précédent :
[JoinNode(7) - [ClassObjectType class=com.sample.Order]], evalCount:100000, elapsedMicro:205274.
Accumulate est évalué pour chaque combinaison $o1/$o2 produite par le JoinNode. Ce qui n'est pas bon.
Cette accumulation sert uniquement à calculer le prix maximum des commandes d'un client.
On ne doit donc l'évaluer qu'une seule fois par client.

Corriger ce défaut, réexécuter le test et identifier le prochain goulot d'étranglement

Apporter une correction pour optimiser le temps global

Atelier 5 : Drools CEP

Importer le projet Maven fourni

Inspecter le code et le comprendre

Écrire une règle qui détecte un arrêt cardiaque :
Si il n'y a pas de battement de cœur dans une fenêtre glissante de 5 secondes

Exécuter les tests, ils devraient détecter un évènement ***HeartAttackEvent***

TP6 : Dépôt de connaissance Maven et KieScanner

Dans cet exercice, nous allons reprendre les Tps précédents.

Réorganisation des sources avec Maven

Nous réorganisons les sources en 2 projet Maven :

- ***6_Rules*** : Contient les classes modèles et les règles. Et les tests
- ***6_Application*** : Contient une classe principale qui scanne le repository Maven toutes les 10s pour récupérer le kjar du premier projet. Ce projet est dépendant de kie-api, drools-compiler, drools-decisiontables et kie-ci

Construction

Construire le projet *6_rules* et le déployer dans le dépôt local Maven
mvn install

Dans le projet 6_application, exécuter les règles

Mise à jour dynamique

Effectuer un changement dans les règles, déployer et observer le chargement dynamique de la base de connaissance

Atelier 7: KieServer et Workbench

7.1. Installation kie Server

Télécharger Wildfly 14 et dézipper

Télécharger une distribution de KieServer et la placer dans :
WILDFLY_HOME/standalone/deployments

Ajouter un utilisateur avec le rôle kie-server :
WILDFLY_HOME/bin/add-user.[sh|bat] -a -u 'kieserver' -p 'kieserver1!' -ro 'kie-server'

Démarrer le serveur en configuration full
WILDFLY_HOME/bin/standalone.[sh|bat] -c standalone-full.xml

Vérifier l'URL :
http://SERVER:PORT/CONTEXT/services/rest/server/

Accéder à la documentation swagger docs et essayer quelques appels REST

7.2. Déploiement du Projet règle via le workbench

7.2.1 Installation du workbench

Télécharger le workbench et le déployer en le plaçant dans :
WILDFLY_HOME/standalone/deployments

2.1 Importing assets
Ajouter un nouveau projet nommé « Asssurance -DRL» et importer les assets suivants en faisant bien attention au nom des packages :

- *Assurance.drl*
- Les 2 classes modèle

Construire et déployer sur le serveur KIE

7.3. Projet client

Dans l'IDE Eclipse, créez un projet qui invoque le moteur Drools et le conteneur Insurance-drl