

## Ficha de Trabalho nº 7

### Sistema Pericial: sistema de incêndios

#### 1. Introdução

O objectivo desta aula consiste na criação de um sistema pericial que detecte a existência de incêndios numa casa equipada com alarmes e aspersores. A memória de trabalho é alterada pelas próprias regras, o que implica que outras regras sejam disparadas.

#### 2. Detalhes do Problema

Numa casa existem várias divisões, cada uma delas pode ter ou não aspersor para combater os incêndios. Existe um alarme geral para toda a casa. Só é possível apagar um fogo numa divisão se esta possuir aspersor. O alarme dispara quando detecta pelo menos um fogo. O alarme desliga quando todos os fogos forem extintos.

##### 2.1 Classes Java

Uma casa possui várias divisões (Classe **Divisao**). Cada divisão possui um nome e pode ter ou não um aspersor (Classe **Aspersor**). A casa tem um alarme (Classe **Alarme**) que dispara mal um fogo (Classe **Fogo**) seja detectado e apenas é desligado quando não houver nenhum fogo na casa.

##### 2.2 Sistema pericial

O Sistema Pericial deve analisar as condições do ambiente numa divisão de uma casa e em caso de detectar algum fogo liga o alarme e os aspersores de água dessa divisão. Após ligar o aspersor de uma divisão, o fogo dessa divisão é extinto. O Sistema Pericial apenas dá indicação de alarme desligado, quando já não existir qualquer fogo.

### 3. Tarefas a executar

#### 3.1 Crie um novo projeto Drools de nome **Fogo1**

#### 3.2 Implemente as seguintes classes

```
public class Divisao {
    String nome;

    //constructor here
    // getter and setter methods here
}

public class Aspersor {
    Divisao divisao;
    boolean ligado;

    //constructor here
    // getter and setter methods here
}

public class Fogo {
    Divisao divisao;

    //constructor here
    // getter and setter methods here
}

public class Alarme {
}
```

#### 3.3 Implemente os construtores, os *getters* e os *setters* para as classes anteriores (excepto **Alarme**);

#### 3.4 No ficheiro de regras implemente as seguintes regras descritas em *pseudocodigo*

##### NOTAS sobre as regras:

- A prioridade de uma regra é atribuída com a instrução **salience**. Quanto mais alto o valor, maior prioridade.
- Algumas instruções para implementar as regras:
  - Analisar uma divisão **\$d: Divisao()**
  - Verificar existência de fogos: **exists Fogo()**
  - Verificar ausência de fogos: **not Fogo()**
  - Ligar o alarme: **insert(new Alarme());**
  - Verificar se o alarme está ligado: **\$a: Alarme()**
  - Verificar se o alarme está desligado: **not Alarme()**
  - Desligar o alarme: **retract(\$a);**
  - Para atualizar o estado da memória use a instrução **update** -> **update(\$a);**
  - Para apagar factos da memória use a instrução **retract** -> **retract(\$f);**

```
rule "Se há fogo numa divisão, ligar aspersor de água"
    prioridade 50
    se
        Analisa divisão
        Existe fogo na divisão
        O aspersor está desligado
    então
        Liga aspersor
        Atualiza estado do aspersor
        Imprime "Aspersor ligado na divisão "
        Fogo apagado
        Imprime "Fogo foi extinto na divisão "
    end
```

**rule "Fogo apagado numa divisão, desligar aspersor"**

```
prioridade 40
se
    Analisa divisão
    O aspersor está ligado
    Não existe fogo na divisão
então
    Desliga aspersor
    Atualiza estado do aspersor
    Imprime "Aspersor desligado na divisão "
End
```

**rule "Liga alarme se há fogos"**

```
prioridade 100
se
    Há fogo
então
    Liga alarme
    Imprime "Alarme ligado"
end
```

**rule "Desliga alarme se não há fogos "**

```
prioridade 10
se
    Não há fogo
    Alarme ligado
então
    Desliga alarme
    Imprime "Alarme desligado"
end
```

**rule "Tudo OK"**

```
prioridade 1
se
    Alarme desligado
    Aspersores desligados
então
    Imprime "Tudo OK"
end
```

3.5 Insira os seguintes factos na memória de trabalho, na função *main*:

```
Divisao d1 = new Divisao ("cozinha");
Divisao d2 = new Divisao ("escritório");
Divisao d3 = new Divisao ("sala");
kSession.insert( d1 );
kSession.insert( d2 );
kSession.insert( d3 );
Aspersor asp1 = new Aspersor( d1, false ); //cozinha tem aspersor
Aspersor asp2 = new Aspersor( d2, false ); //escritório tem aspersor
//...                                     // A sala não tem aspersor
kSession.insert( asp1 );
kSession.insert( asp2 );

Fogo f1 = new Fogo (d1); //Fogo na cozinha
Fogo f2 = new Fogo (d2); //Fogo no escritório
Fogo f3 = new Fogo (d3); //Fogo na sala
kSession.insert( f1 );
kSession.insert( f2 );
kSession.insert( f3 );
kSession.fireAllRules();
```

- 3.6 Teste o Sistema Pericial. Como verifica as regras “Tudo OK” e “Desliga alarme se não há fogos ” nunca são executadas, visto que o fogo na sala, por ausência de aspersor nunca é extinto. Coloque o código (na função **main**) que cria um aspersor na sala e teste o SP. Verifique que, quando todos os fogos são extintos, as regras anteriormente referidas já são executadas.
- 3.7 As regras anteriores efetuam a inserção e remoção do alarme de forma explícita, utilizando as instruções **insert()** e **retract()**. É possível inserir factos na memória de trabalho de uma forma alternativa utilizando a instrução **insertLogical()**. Neste caso, a presença do facto na memória de trabalho depende da verdade da regra que foi ativada. Enquanto a condição da regra for verdadeira, o facto mantém-se na memória. Assim que a condição se torna falsa, o facto é automaticamente retirado não sendo necessário recorrer à instrução **retract()**.  
Efetue as alterações necessárias no código anterior de modo a garantir que a inserção do alarme na memória (i.e., a indicação que o alarme está ligado) seja feita com a instrução **insertLogical()**.
- 3.8 Assuma que as características de uma casa se encontram **sumariadas num ficheiro de texto**. O ficheiro possui uma linha de cabeçalho, seguida de várias linhas. Cada linha do ficheiro representa uma divisão, e para cada divisão aparece o nome, a indicação se existe fogo e se possui aspersor.

```
Divisao;aspersor?;Fogo?  
cozinha;sim;sim  
escritorio;sim;sim  
sala;sim;sim  
quarto 1;sim;sim  
quarto 2;sim;sim  
quarto 3;sim;sim  
estufa;sim;sim  
garagem;sim;sim
```

No ficheiro **DroolsTest.java**, altere a forma de inserção dos factos na memória de trabalho, de forma a que toda a informação a inserir seja lida a partir do ficheiro **fogo.txt**

## 4. Exercício adicional

Pretende-se um sistema pericial para atualizar o saldo bancário de diversos clientes.

Na memória de trabalho serão colocados factos representando as contas dos diversos clientes e os respectivos movimentos de levantamento. A inferência deverá associar movimentos a clientes e atualizar os respectivos saldos.

### 4.1 Classes

Criar a classe **Conta** com 2 variáveis: Nome (*String*) e Saldo (*double*). Criar também o construtor, os *getters* e os *setters*.

Criar a classe **Movimento** com 2 variáveis: Nome (*String*) e Montante (*double*). Criar também o construtor, os *getters* e os *setters*.

### 4.2 Inserção de Informação na Memória de Trabalho

Coloque a seguinte informação na memória de trabalho:

- A Ana tem uma conta com 100.0€;
- O Jorge tem uma conta com 250.0€;
- Existe um pedido de levantamento para a conta da Ana de 10.0€;
- Existe um pedido de levantamento para a conta do Jorge de 100.0€;
- Existe um pedido de levantamento para a conta da Ana de 150.0€;
- Existe um pedido de levantamento para a conta do Luís de 75.0€;

### 4.3 Criação de Regras

Regra 1: Detecção de pedidos de levantamento sem saldo

Apresenta uma mensagem de alerta para todos os pedidos de levantamento que não tenham saldo na respectiva conta

Regra 2: Processa levantamento

Atualiza o saldo das contas que tenham um pedido de levantamento que possa ser satisfeito. Apresenta uma mensagem a informar que o processo foi bem-sucedido.

Regra 3: Escreve Saldos Finais

Apresenta o saldo final das contas depois de todos os pedidos de levantamento terem sido processados.

### 4.4 Teste

Após terminar a implementação, teste o sistema pericial e verifique se o resultado é o pretendido.

### 4.5 Melhoramentos

Crie uma regra que aplique uma taxa de juro de 5% a todos os depósitos e adicione-a ao sistema implementado no ponto anterior. Esta regra só deverá ser ativada depois de todos os pedidos de levantamento terem sido processados.

Após terminar a implementação, teste o sistema pericial e verifique se o resultado é o pretendido.