Ficha de Trabalho nº 6

Sistema Pericial: atribuição de crédito bancário

Sintaxe das regras DRL

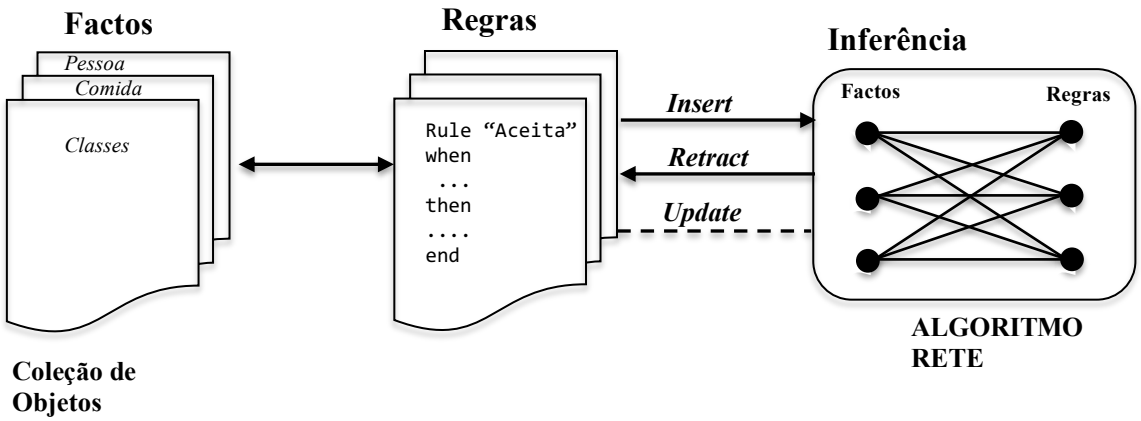
1. Introdução

Um sistema pericial (SP) consiste num sistema baseado em conhecimento cujo objetivo é simular a perícia humana num domínio específico. Um SP é constituído por um conjunto de **factos** e **regras** sobre o domínio (tal como existem no especialista humano) que permitem oferecer respostas e sugestões aos seus utilizadores. Os SP são usados em vários domínios: diagnóstico médico, análise de risco no sector bancário ou segurador, controle de sistemas, etc.

2. Drools

Nesta aula vamos implementar um SP muito simples usando a ferramenta Drools (usada através do Eclipse). Neste SP pretende-se representar conhecimento sobre os clientes de um banco e as regras que o banco usa para decidir se aceita ou recusa um pedido de crédito de um cliente.

No Drools, a representação do conhecimento e as inferências sobre o mesmo necessitam dos **factos**, de **regras** e de conteúdo na **memória de trabalho**. O mecanismo de inferência define como as regras atuam sobre o conhecimento (algoritmo RETE):



2.1 Coleção de Objetos

No Drools os objetos são definidos através de Java, usando **classes**. Por exemplo, para guardar informação sobre uma pessoa podemos ter a classe:

```
public class Pessoa{
      private String nome;
      private int idade;
      private char sexo;
      //construtor, getters, setters
}
```

2.2 Regras DRL

As regras DRL têm a seguinte sintaxe:

Formato DRL:

```
rule <nome da regra>
Atributos da regra
#comentário ou
// comentário ou
/* comentário */
when
     <condições (LHS)>
then
     <ações (RHS)>
end
```

Exemplo

```
rule "permite crédito"
salience 100 //prioridade
#ver se pessoa tem mais de 18 anos
when
     $p:Pessoa(idade >= 18)
then
     System.out.println("Permite");
end
```

Nas condições (LHS) podem usar-se os seguintes elementos:

(O bloco **`when`** devolve **`true`** quando todas as expressões nele contidas são avaliadas com **`true`**.)

Elemento	Significado
()	Agrupar
&&	AND
,	AND, baixa prioridade
||	OR
>, <, >=, <=, ==, !=	Comparação
contains	
exists	
memberof	
not memberof	
matches	
eval	Avaliar uma instrução Java
:	Ligação aos dados (atribuição)

Na ligação aos dados usam-se expressões contendo os elementos anteriores. Vejamos alguns exemplos e respetivo significado:

Exemplo	Significado
`Pessoa()`	Devolve todos os objetos do tipo Pessoa na memória de trabalho
`$p: Pessoa()`	Retorna todos os objetos do tipo Pessoa. Faz a ligação dos objetos devolvidos com a variável $p
`$p: Pessoa(idade >= 18)`	Retorna todos os objetos do tipo Pessoa cuja idade seja maior ou igual a 18
`$p: Pessoa(idade >= 18, sexo == 'M')`	Retorna todos os objetos do tipo Pessoa cuja idade seja maior ou igual a 18 e o sexo masculino
`$p: Pessoa(idade >= 18 || sexo == 'M')`	Retorna todos os objetos do tipo Pessoa cuja idade seja maior ou igual a 18 ou o sexo masculino
`Pessoa( $idadeF : idade, sexo == 'F' )` `Pessoa( idade == ( $idadeF + 2) , sexo == 'M' )`	Retorna os homens com dois anos a mais que cada mulher

Nas acções (RHS) podem usar-se os seguintes elementos:

- Código Java
- Funções de alteração da memória de trabalho:

  `update` Avisa que algum objeto mudou e as regras precisam ser reconsideradas

  `insert` Cria um novo objeto na memória de trabalho

  `insertLogical` Mesmo que o anterior, mas remove o objeto quando não existirem mais factos que suportem a regra que o criou

  `retract` Remove um objeto da memória de trabalho

3. Implementação do SP em Drools

3.1 Exemplo: atribuição / recusa de crédito

Para o exemplo da aula de hoje (atribuição de crédito bancário) vamos:

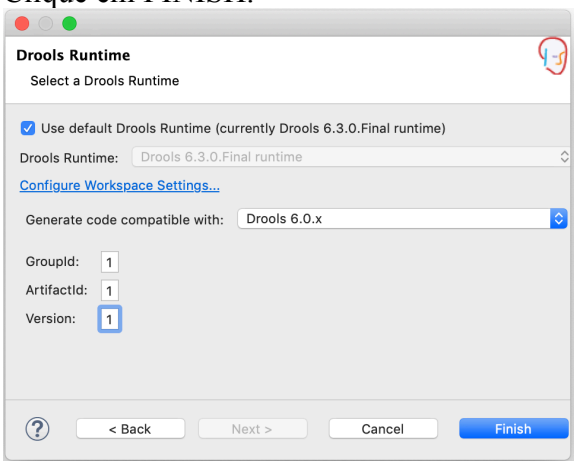
1) Definir os factos (no Drools faz-se usando a linguagem Java)
   Classe Pessoa
   i. nome
   ii. idade

2) Definir as regras
   Se idade >= 18 **Então** Atribuir Crédito
   Se idade < 18 **Então** Recusar crédito

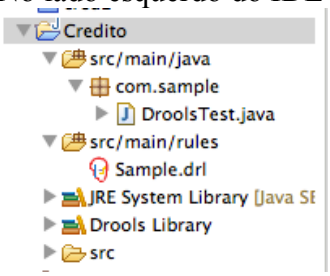
3) Criar memória de trabalho:
   a. Criar pessoa Rafael, 20 anos, inserir na memória
   b. Criar pessoa Maria, 15 anos, inserir na memória

3.2 Execute os seguintes passos

- Inicie o Eclipse e faça: **`File - New Project - Drools Project`**
- Atribua o nome ao projeto (Credito1)
- Selecione as 2 primeiras *check box*
- Clique em NEXT.
- Active a Check box "**`Use default drools Runtime`**" (verifique se a diretoria está correta)
- Preencha os campos **GroupID**, **ArtifactID**, e **Version** com um caracter, por exemplo o dígito **'1'**.
- Clique em FINISH.



- No lado esquerdo do IDE deve surgir a seguinte estrutura:



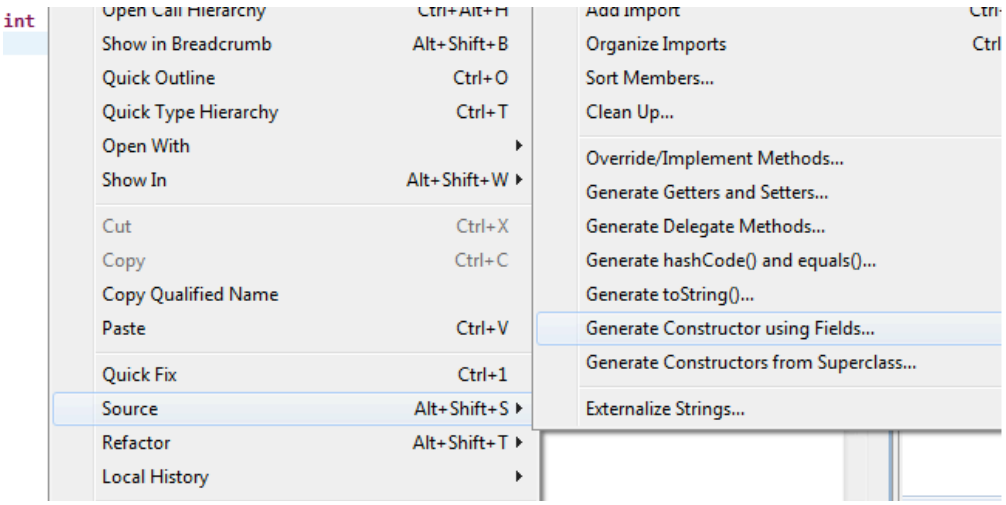
Definir os factos (`Class Pessoa`):

- Com o botão direito do rato sobre **`com.sample`** faça: `New-Class`
  - No campo `Name` escreva: Pessoa
  - Clique em FINISH
  - Do lado esquerdo do IDE deve surgir o ficheiro `Pessoa.java`
  - Complete o código com:

```
package com.sample;
public class Pessoa{
      private  String nome;
      private  int idade;
}
```

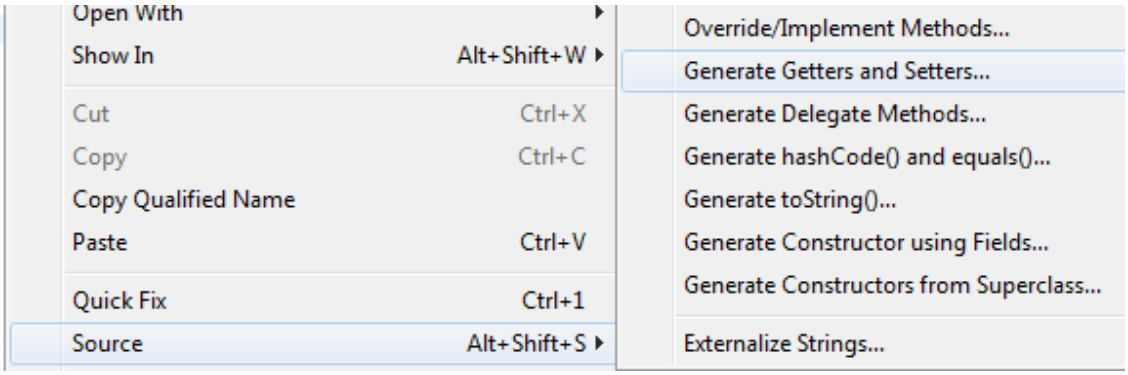
  - O Drools possui uma forma automática de criar o código para o construtor, getters e setters.

    Construtor:
    - Coloque o cursor na linha antes da chaveta de fecho e faça (veja Figura):

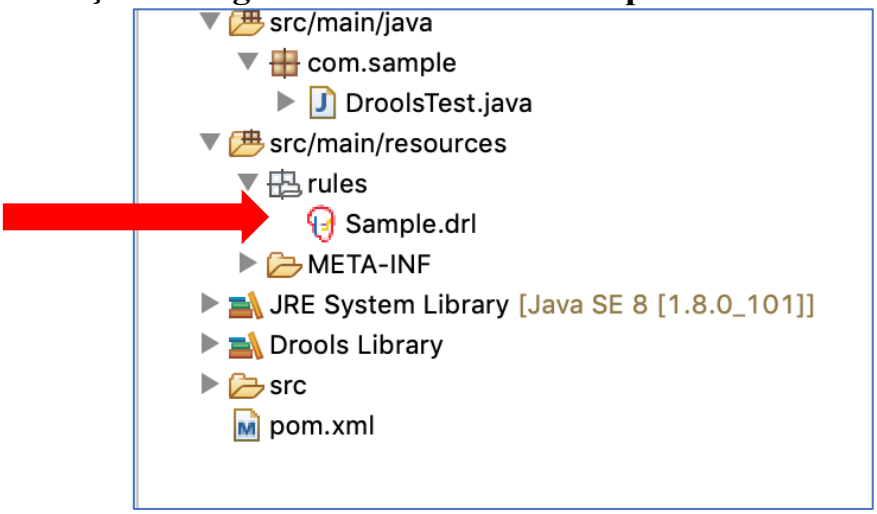


      - Botão direito do rato
      - Escolha `Source`
      - Escolha `Generate Constructor Using Fields`
      - Selecione os dois campos nome e idade e complete a operação.
      - O código do construtor deve surgir dentro da classe `Pessoa`.

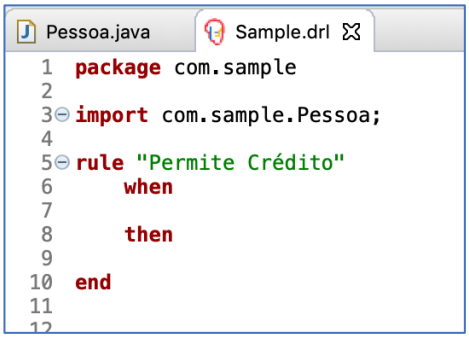
    Getters (não são necessários os setters neste exemplo)
    - Coloque o cursor na linha antes da chaveta de fecho e faça
      - Botão direito do rato
      - Escolha `Source`
      - Escolha `Generate Getters and Setters`
      - Expanda os campos nome e idade e selecione apenas as check boxes relativas a `getIdade()` e `getNome()`. Complete a operação com OK,
      - O código dos dois getters deve surgir dentro da classe `Pessoa`.



- **Na secção de regras edite o ficheiro Samples.drl fazendo duplo clique.**



- Edite/altere o ficheiro e deixe o código como está na figura abaixo:



- A estrutura de uma regra é a seguinte:

```
rule “Nome da Regra”
when
        <condições>
then
        <ações>
end
```

- Escreva as duas regras para atribuição/não atribuição de crédito:

```
package com.sample
import com.sample.Pessoa;

rule "Permite Crédito"
	when
		$p : Pessoa(idade >= 18)
	then
		System.out.println("Crédito Permitido para " + $p.getNome() );
end

rule "Crédito Negado"
	when
		Pessoa( $nome : nome, idade < 18)
	then
		System.out.println("Crédito Negado para " + $nome);
end
```

Colocar Factos (pessoas) na memória de trabalho

- Para executar esta tarefa vamos aproveitar o código gerado automaticamente pelo Drools.
- Faça duplo clique sobre o ficheiro `DroolsTest.java`
  Faça as seguintes alterações:
  - Acrescente `import com.sample.Pessoa;`

```
1  package com.sample;
2  import com.sample.Pessoa;
3  import org.kie.api.KieServices;
4  import org.kie.api.runtime.KieContainer;
5  import org.kie.api.runtime.KieSession;
6
7  /**
8   * This is a sample class to launch a rule.
9   */
10 public class DroolsTest {
11
12     public static final void main(String[] args) {
```

  - Na função `main` apague as três linhas após o comentário `//go!` Referentes à criação de objetos `Message`
  - Crie duas instâncias do da classe Pessoa:

```
Pessoa p1 = new Pessoa("Rafael", 25);
Pessoa p2 = new Pessoa("Maria", 15);
```

  - Para colocar esta informação na memória de trabalho deve usar-se o método `insert` da variável `ksession`:

```
kSession.insert(p1);
kSession.insert(p2);
```

  - Para ativar o SP e executar as regras deve usar-se o método `fireAllRules`:

```
kSession.fireAllRules();
```

Deverá obter o seguinte código:

```
public class DroolsTest {

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

            // go !
            Pessoa p1 = new Pessoa("Rafael", 25);
            Pessoa p2 = new Pessoa("Maria", 15);

            kSession.insert(p1);
            kSession.insert(p2);

            kSession.fireAllRules();

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


}
```

- Execute o programa
  - `Run – Run As – Java Application`
- Na consola deve surgir o seguinte resultado:

```
Crédito negado para Maria
Crédito permitido para Rafael
```

3.3 Melhorar o sistema pericial

Faça as seguintes melhorias ao seu SP:

- **Factos:**
  - Crie uma nova classe `Comida` com dois campos:

```
Class Comida{
      private String comida;
      private int calorias;
}
```

    Dentro da classe gere o **construtor** e os **getters** de forma automática (veja como fez anteriormente)

  - Na classe `Pessoa` adicione mais um campo:

```
public class Pessoa{
      private String nome;
      private int idade;
      private String comidaPref;
}
```

- Na classe `Pessoa` altere o construtor para incluir o novo campo e adicione o *getter* para o novo campo.

- **Regras:**
  - Acrescente o `import` da classe `Comida` **: *import com.sample.Comida;***

  - **Altere a regra** "Permite Crédito":
    - Se a pessoa tiver pelo menos 18 anos **E** se a comida preferida tiver menos de 500 calorias
    - Então: Atribui crédito
      - "*Pessoa maior de idade, com alimentação saudável. Crédito permitido para <nome>*"

```
rule "Permite Crédito"
    when
        $c:Comida()
        $p:Pessoa(idade>=18, comidaPref==$c.getComida(), $c.getCalorias()<=500)
    then
        System.out.println("Pessoa maior de idade, com alimentação saudável.
        Crédito permitido para  "+ $p.getNome());
end
```

  - **Crie uma nova regra** "Nega Crédito 2"
    - Se a pessoa tiver pelo menos 18 anos **E** a comida preferida tiver mais de 500 calorias
    - Então: Recusa crédito
      - "*Pessoa maior de idade, com alimentação perigosa. Crédito negado para <nome>*"

```
rule "Nega Crédito 2"
  // COMPLETAR
end
```

  - **Crie uma nova regra** "Não sabe"
    - Se a pessoa tiver pelo menos 18 anos **E** a comida preferida for desconhecida (use a instrução **`forall`** para testar todas as comidas presentes na memória)
    - Então: Imprime a mensagem:
      - *"Informação insuficiente sobre alimentação de <nome>"*

```
rule "Não sabe"
    when
       $p:Pessoa($nome: nome, idade >= 18)
       forall($c:Comida(comida!=$p.getComidaPref()))
    then
        System.out.println("Informação insuficiente sobre alimentação de " +
        $nome);
     end
```

- **Memória:**
  - No ficheiro `DroolsTest.java` faça o **`import com.sample.Comida;`**
  - E na função **`main`** crie as seguintes instâncias e coloque-as na memória (altere o código relativo às pessoas):

```
Comida enchido = new Comida("Enchidos", 1000);
Comida salada = new Comida("Saladas", 100);
Pessoa p1 = new Pessoa("Rafael", 25, "Saladas");
Pessoa p2 = new Pessoa("Maria", 15, "Enchidos");
Pessoa p3 = new Pessoa("David", 37, "Enchidos");
Pessoa p4 = new Pessoa("Daniela", 12, "Saladas");
Pessoa p5 = new Pessoa("Jorge", 45, "Pizzas");
kSession.insert(enchido);
kSession.insert(salada);
kSession.insert(p1);
kSession.insert(p2);
kSession.insert(p3);
kSession.insert(p4);
kSession.insert(p5);
```

  - Corra a aplicação e confirme se obtém o seguinte resultado na consola:

```
Aprovado para Rafael – alimentacao saudavel
Recusado para Daniela – menor de idade
Recusado para Maria – menor de idade
Recusado para David – alimentação perigosa
Informação insuficiente sobre a alimentação de Jorge
```

- Na classe `Pessoa` adicione mais um campo:

```
public class Pessoa{
	private String nome;
	private int idade;
	private String comidaPref;
	private double rendimento;
}
```

- Na classe `Pessoa` altere o construtor para incluir o novo campo e adicione o *getter* para o novo campo.

- **Altere/Acrescente as Regras:**

  - **<u>Altere a regra</u>** "Permite Crédito":
    - Se a pessoa tiver pelo menos 18 anos **E** se a comida preferida tiver menos de 500 calorias **E** o rendimento for superior ou igual a 1000
    - Então: Atribui crédito
      - "*Pessoa maior de idade, com alimentação saudável, rendimento acima do limiar 1000. Crédito permitido para <nome>*"

- **<u>Acrescente uma regra</u>** "Nega Crédito3":
  - Se a pessoa tiver pelo menos 18 anos **E** se a comida preferida tiver menos de 500 calorias **E** o rendimento for inferior a 1000
  - Então: Nega crédito
    - "*Pessoa maior de idade, com alimentação saudável, mas sem rendimento mínimo. Crédito negado para <nome>*"

Memória:

- No ficheiro `DroolsTest.java` na função **`main`** crie as seguintes instâncias e coloque-as na memória (altere o código relativo às pessoas):

```
Comida enchido = new Comida("Enchidos", 1000);
Comida salada = new Comida("Saladas", 100);
Pessoa p1 = new Pessoa("Rafael", 25, "Saladas", 1500);
Pessoa p2 = new Pessoa("Maria", 15, "Enchidos", 1500);
Pessoa p3 = new Pessoa("David", 37, "Enchidos", 500);
Pessoa p4 = new Pessoa("Daniela", 12, "Saladas", 0);
Pessoa p5 = new Pessoa("Jorge", 45, "Pizzas", 1200);
Pessoa p6 = new Pessoa("Salvador", 45, "Saladas", 200);

kSession.insert(enchido);
kSession.insert(salada);
kSession.insert(p1);
kSession.insert(p2);
kSession.insert(p3);
kSession.insert(p4);
kSession.insert(p5);
kSession.insert(p6);
```

- Corra a aplicação e confirme se obtém o seguinte resultado na consola:

```
Aprovado para Rafael – alimentacao saudavel e rendimento >= 1000
Recusado para Daniela – menor de idade
Recusado para Maria – menor de idade
Recusado para David – alimentação perigosa
Recusado para Salvador – alimentação saudável, mas sem rendimento mínimo
Informação insuficiente sobre a alimentação de Jorge
```