

DROOLS

Direto ao ponto: versão 7.7.0.Final

```
package droolstutorial

rule "Rules with Drools"
    when
        //conditions
    then
        //actions
        System.out.println("Hello World!!");
    end
```



@Author
João Marlon Souto Ferraz

Sumário

Introdução	3
Overview	4
1. Tutorial do Drools	6
1.1. Instalando Ferramentas do Drools	6
1.1.1. Criando o primeiro projeto	10
1.2. Modelo de dados utilizado no tutorial	13
1.3. Core: Projeto do modelo de dados	14
1.3.1. Primeiro elemento da linguagem Drools	19
1.3.2. Adicionando uma condição simples	20
1.3.3. Usando variáveis globais	22
1.3.4. Usando call-backs para registrar as atividades no Drools Runtime ...	24
1.3.5. Quando e como uma regra é executada	28
1.4. Binding: Relacionando fatos e atributos em uma regra	32
1.4.1. Caso de Teste	36
1.4.2. Ligação dos Fatos	36
1.4.3. Ligação de Atributos	37
1.4.4. Calculando o balanço	40
1.5. Drools language: Exemplos de uso.	43
1.5.1. Constraint	46
1.5.2. Acessor	47
1.5.3. And/or	48
1.5.4. Not	49
1.5.5. Exists	50
1.5.6. ForAll	51
1.5.7. From	52
1.5.8. Collecting	53

1.5.9. Accumulating	55
1.6. Fluxo de regras: organizando a execução de regras em projetos maiores 58	
1.6.1. Criando seu primeiro fluxo de regras	59
1.6.2. Iniciando um fluxo de regras a partir de uma regra.....	64
1.6.3. Fluxo de regras com condições	65
2. Tutorial BRMS.....	71
2.1.1. Instalando e configurando kie-wb e kie-server.....	73
2.1.2. Business Central para Kie Drools Workbench	78
1.1.1. Kie Drools Workbench	79
1.1.1.1. Importando os fatos.....	83
1.1.1.2. Criando fluxo de regras	85
1.1.1.3. Criando arquivo de regras para o fluxo.....	88
1.1.1.4. Criando cenário de testes.....	89
1.1.1.5. Executando regra do WB pelo Eclipse;	94
1.1.2. Modelo de arquitetura	99
Referências	102

Introdução

Este tutorial é para os recém-chegados ao Drools, ele é voltado para a versão 7.7.0.Final.

O dia a dia é regido por regras. Há, por exemplo, a regra de parar no sinal vermelho e a regra de que só pode tirar a carteira de habilitação quando completar 18 anos, apesar de muitos não as seguirem, essas regras não deixam de existir. No ambiente corporativo não é diferente: há diferentes regras para diferentes cenários, que precisam ser cumpridos para que a estrutura planejada pela organização se comporte como esperada.

Como apresentado na documentação oficial do Drools, ao usá-lo, você mudará o paradigma de desenvolvimento clássico normal de desenvolvimento, indo de procedural para Programação Declarativa. Neste paradigma, você pode expressar a lógica de um programa sem ter que descrever explicitamente o fluxo de instruções que devem ser seguidas. Isso significa que o controle do fluxo não é determinado pela ordem das regras, nem pela ordem dos dados recebidos, mas pelas condições declaradas nestas regras, ela permite que qualquer número de regras seja escrito sem se preocupar com o local que o código está.

A mudança não é complexa de realizar e posso assegurá-lo que, ao fazer os exercícios completamente sozinho e sem pular diretamente para as soluções em código, você compreenderá bem o que iremos estudar passo a passo apenas neste tutorial.

Mas por quê usar regras? Essa é a primeira pergunta que fazemos. Você pode ainda estar um pouco confuso sobre por que as regras são algo útil. Se pensarmos nisso, em termos de uma regra ou de poucas, podemos considerar melhor fazê-lo diretamente no código imperativo como o Java, por exemplo. Como desenvolvedores, estamos acostumados dividir os requisitos em uma lista de etapas a serem seguidas, e pronto. No entanto, a força principal por trás das regras de negócios não vem de uma regra ou pequeno grupo de regras, vem de um grande grupo de regras em constante mudança que define um sistema tão complexo que exigiria um trabalho extenso para mantê-lo se o fizéssemos com código regular. Com o crescimento da base de código, muitas regras podem trabalhar juntas para definir sistemas complexos organicamente. Se precisamos implementar novas requisitos, modificar os existentes, substituir parâmetros ou alterar a estrutura do comportamento do sistema de maneiras inesperadas, a única coisa que precisaremos fazer com as regras é implementar as novas regras que agora se aplicam e remover aqueles que não

apliquem mais. Isso é possível porque as regras de negócios funcionam nos seguintes princípios:

- Elas são independentes;
- Elas podem ser facilmente atualizadas;
- Cada regra controla a quantidade mínima de informação necessária;
- Elas permitem que mais pessoas de diferentes origens colaborem;

Overview

Na documentação oficial temos que: o **Drools** é um sistema de gerenciamento de regras de negócios com um mecanismo de regras baseadas em inferência de encadeamento direto e encadeamento reverso, permitindo avaliação rápida e confiável de regras de negócios e processamento complexo de eventos. Um mecanismo de regras também é um bloco de construção fundamental para criar um sistema especialista que, em inteligência artificial, é um sistema de computador que emula a capacidade de tomada de decisão de um especialista humano. Você vai entender melhor com os exemplos, fique tranquilo.

Os desenvolvedores estão acostumados a uma abordagem processual quando se trata de implementar os requisitos de negócios do software. Por consequência, a implementação dessas regras depende da competência e do entendimento que o desenvolvedor tem para implementá-las – o que, para muitos, é uma limitação claramente identificada. Também sabemos que a programação orientada a objetos pode ser vista como “a solução” para a implementação e agilidade dos negócios, e muitos desenvolvedores confiam nesse paradigma. Porém, no final, a abordagem orientada a objetos acaba com o clássico “código de espaguete”. Para evitar esse final, foi adotada uma nova abordagem denominada “programação declarativa”: esta é uma grande mudança e vamos tentar nos concentrar nisso ao aprender as ferramentas em torno do Drools.

Na primeira parte o Drools será apresentado para você. Como configura-lo no eclipse, escrever um fato, uma regra e fazer a brincadeira acontecer. Será uma parte muito importante para conhecer um pouco da linguagem desta ferramenta. Nela, também, você conhecerá um pouco do **jBPM**: um conjunto flexível de Gerenciamento de Processos de Negócios, do inglês Business Process Management, que permite modelar seus objetivos de negócios, descrevendo as etapas que precisam ser executadas para atingir esses objetivos.

Na segunda parte aprenderá como configurar o Drools Workbench: um aplicativo da Web com todos os recursos para a composição visual de regras e processos de negócios personalizados, onde serão simulados os mesmos exemplos que forem criados no eclipse, e serão executados via API REST disponibilizados num container do Kie-server.

Todo código apresentado por aqui pode ser encontrado no [github](https://github.com/jmarlonsf/)¹. O título do projeto terá como sufixo “droolstutorial”.

¹ <https://github.com/jmarlonsf/>

1. Tutorial do Drools

O Drools fornece um IDE baseado em Eclipse (que é opcional), mas em seu núcleo apenas o Java 1.5 (Java SE) é necessário.

O uso do plug-in do Eclipse não é obrigatório. Os arquivos de regras são apenas entradas textuais (ou planilhas, conforme o caso) e o IDE (também conhecido como o Rule Workbench) é apenas uma conveniência. As pessoas integraram o mecanismo de regras de várias maneiras, não há "tamanho único".

1.1. Instalando Ferramentas do Drools

Baixe e instale os seguintes itens:

1. Java Virtual Machine versão 8.x+;
2. Eclipse IDE;
3. [Faça o download do Drools Runtime e Tools <clique aqui>²](https://download.jboss.org/drools/release/7.7.0.Final/droolsjbpm-integration-distribution-7.7.0.Final.zip).
4. Descompacte o arquivo baixado em uma pasta fora do workbench do eclipse e fora do projeto. Algo como: "C:\Users\SeuUsuario\Documents\Drools\Unzip-DoArquivoBaixado".

² <https://download.jboss.org/drools/release/7.7.0.Final/droolsjbpm-integration-distribution-7.7.0.Final.zip>

5. Instale o novo software no eclipse apontando para a pasta “org.drools.updatesite”, que está dentro do arquivo que você baixou.
 - Eclipse IDE > Help > Install New Software > Work With > [...] org.drools.updatesite. Selecione Drools and jBPM, clique: Next, Accept the terms. Etc.. e reinicie o Eclipse.

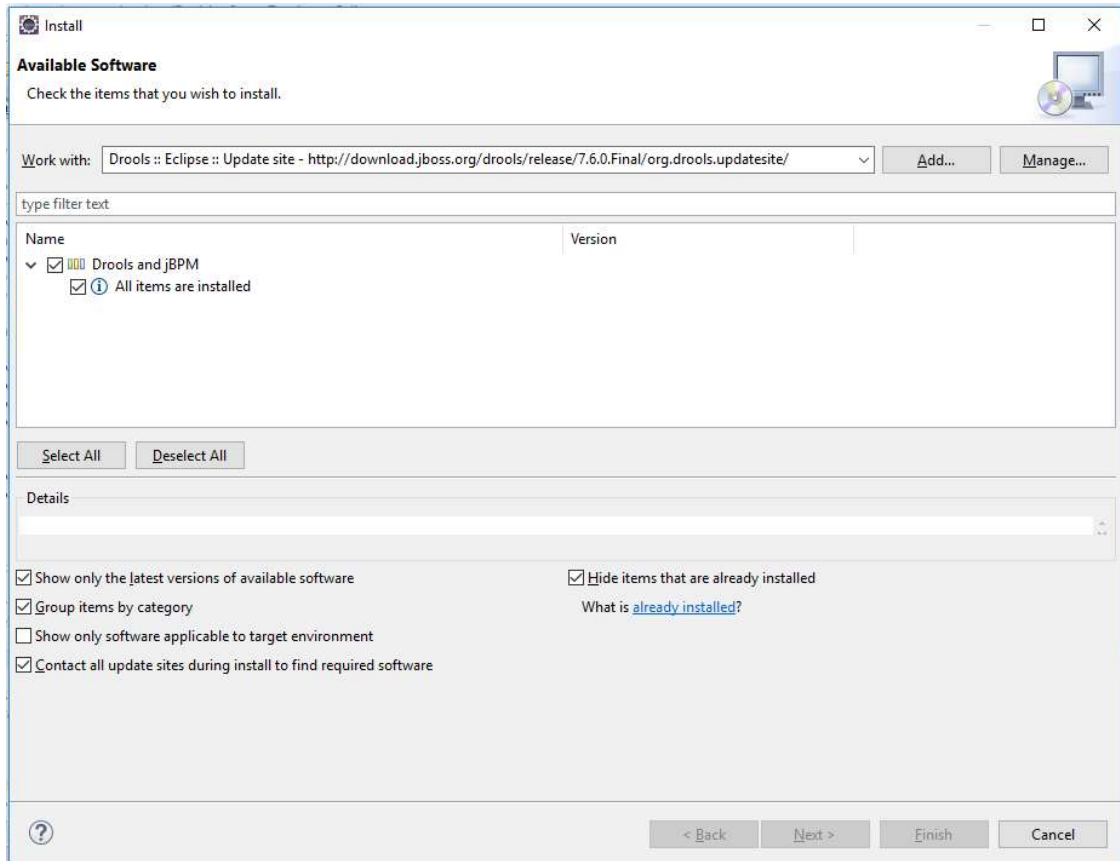


Figura 1: Instalando Drools

6. Configure as versões do Drools Runtimes instaladas.

- Eclipse IDE > Windows > Preferences > Drools > Installed Drools Runtimes.

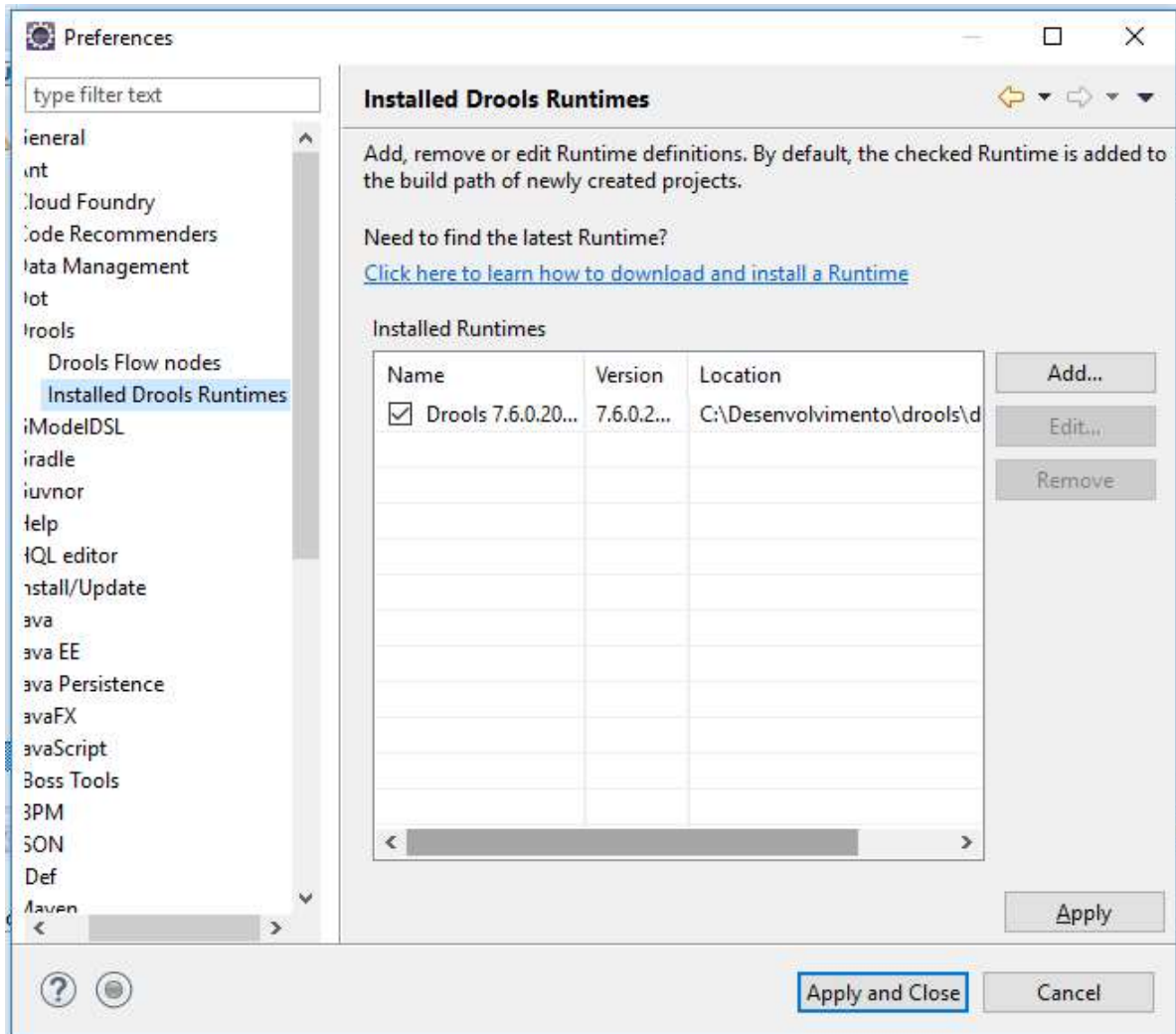


Figura 2: Gestão dos Runtimes instalados

- Clique em Add... > Browse... e navegue até a pasta binaries, que está dentro do arquivo que você baixou. Clique OK > OK > Apply and Close > Reinicie o Eclipse.

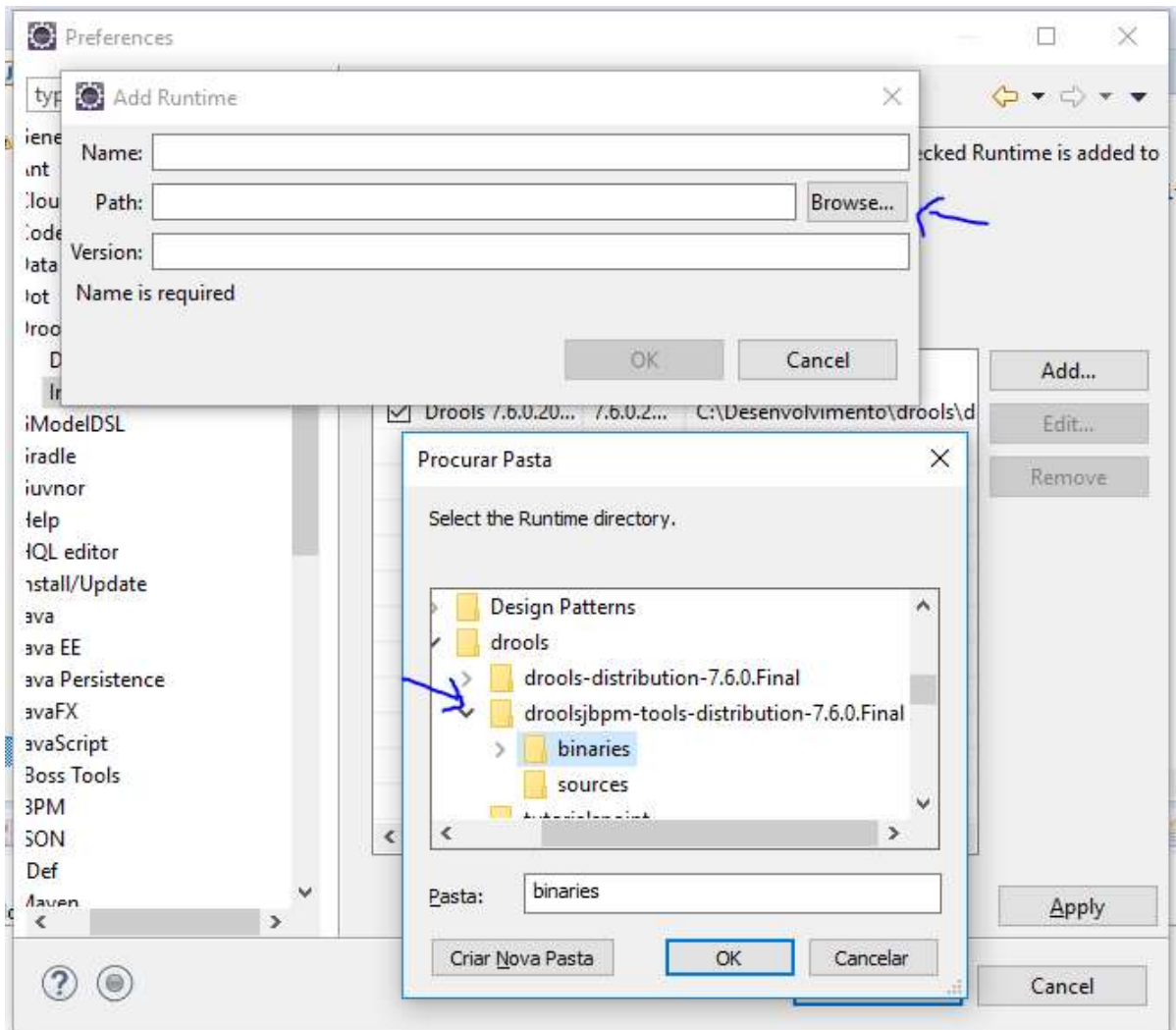


Figura 3: Configurando Runtimes

- Toda vez que você instalar um novo Runtime, o eclipse deve ser reiniciado.

1.1.1. Criando o primeiro projeto

Clique na aba: File > New > Other > Drools > Drools Project > Next.

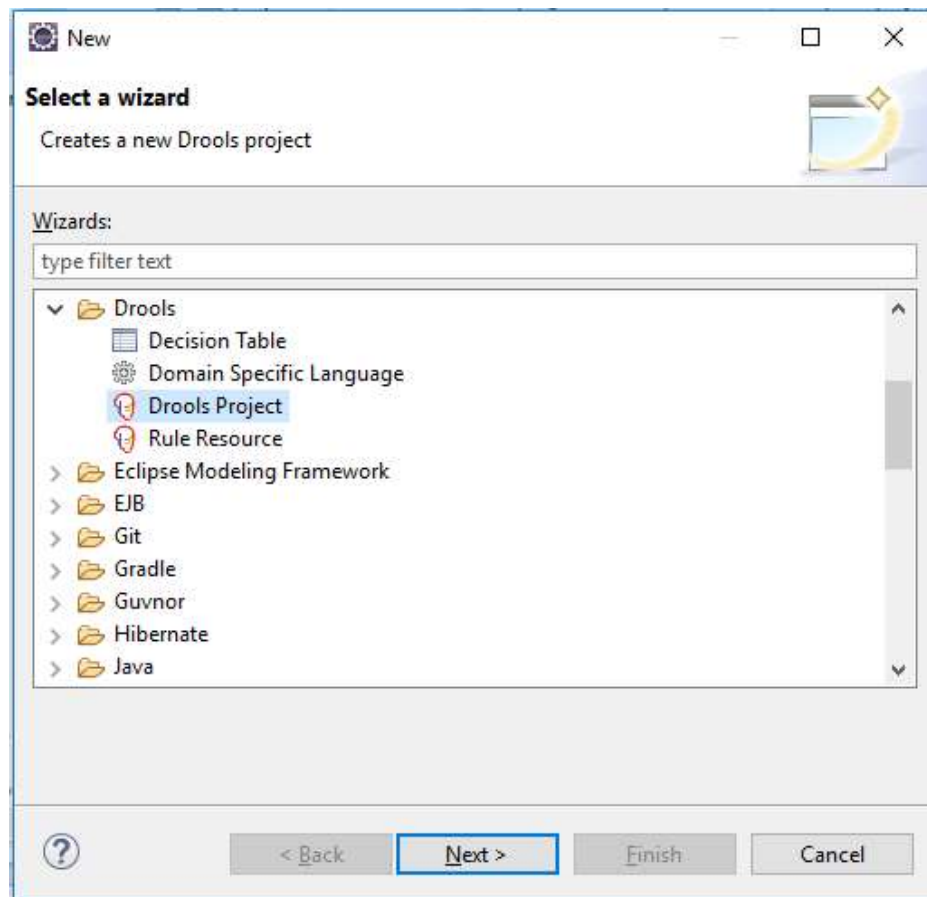


Figura 4: Criando novo projeto Drools

Selecione o botão do meio, o que tem alguns arquivos de exemplos e clique em Next.

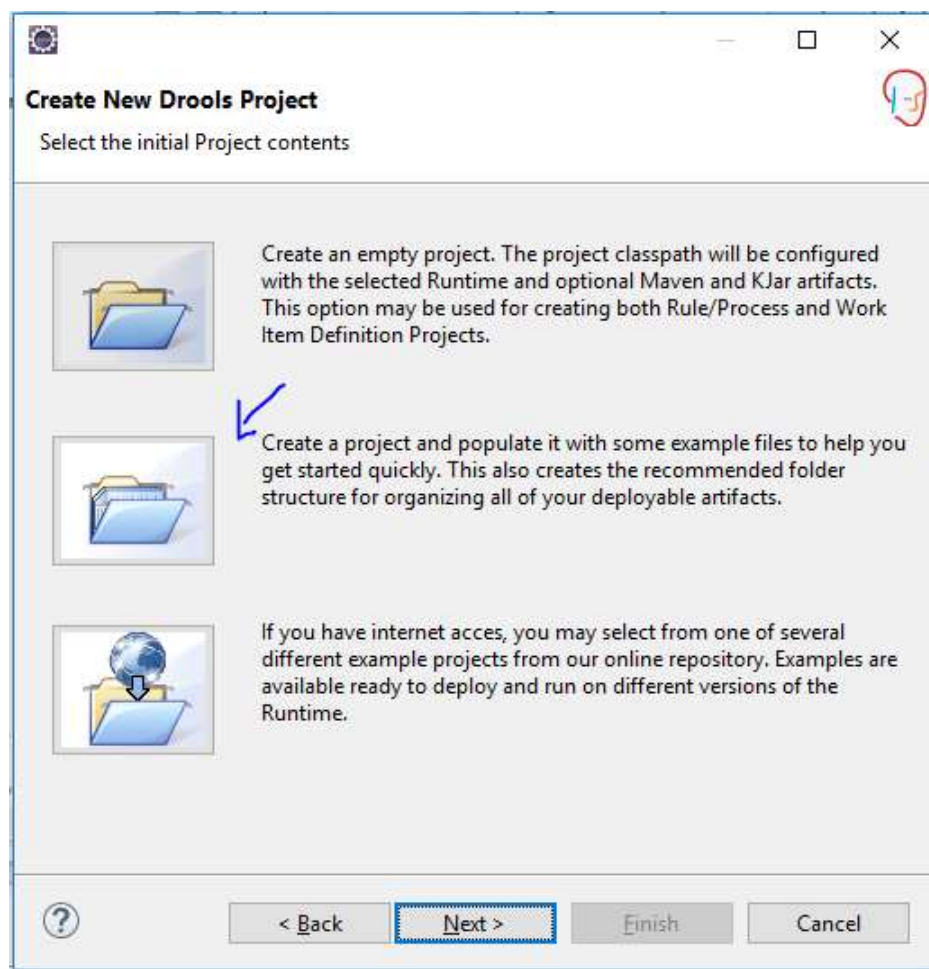


Figura 5: Criando novo projeto com exemplos

Acrescente o nome do projeto e o workspace e clique em Finish.

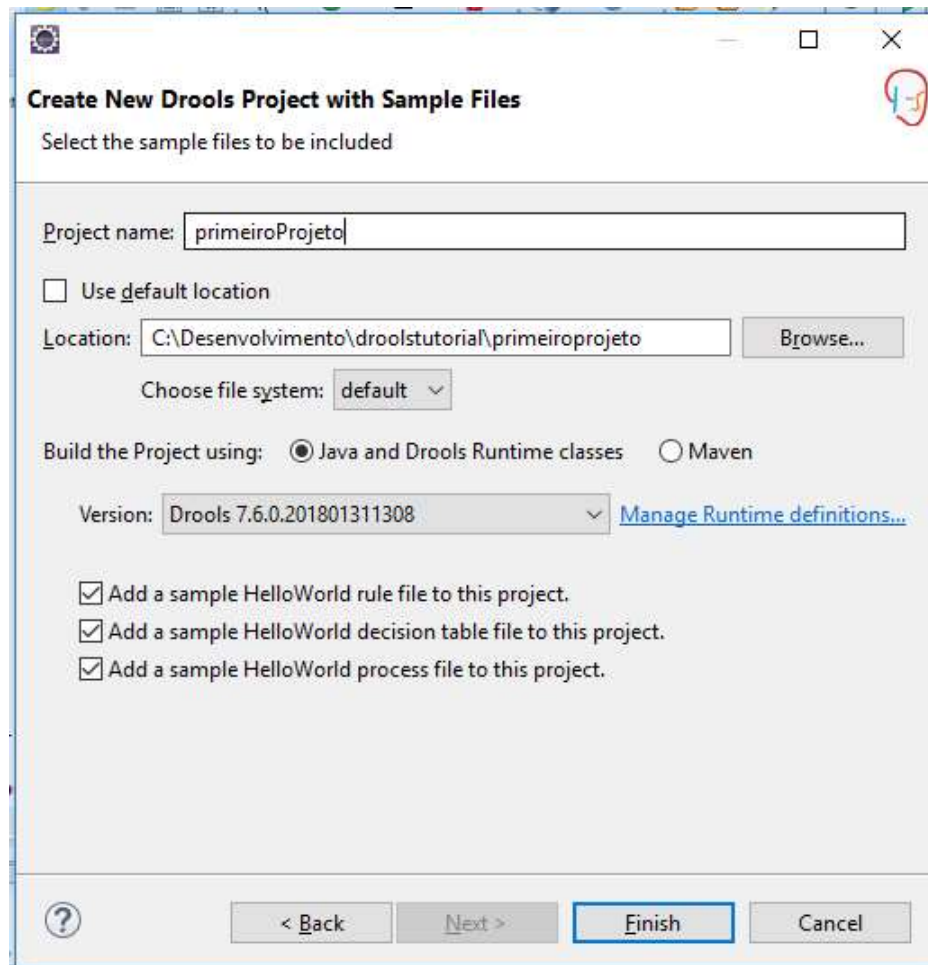


Figura 6: Definindo nome, workspace, versão do Runtime e acrescentando exemplos ao projeto.

Execute a classe DroolsTest.java como Java Application: Se aparecer a mensagem abaixo no console, então a instalação foi realizada com sucesso!

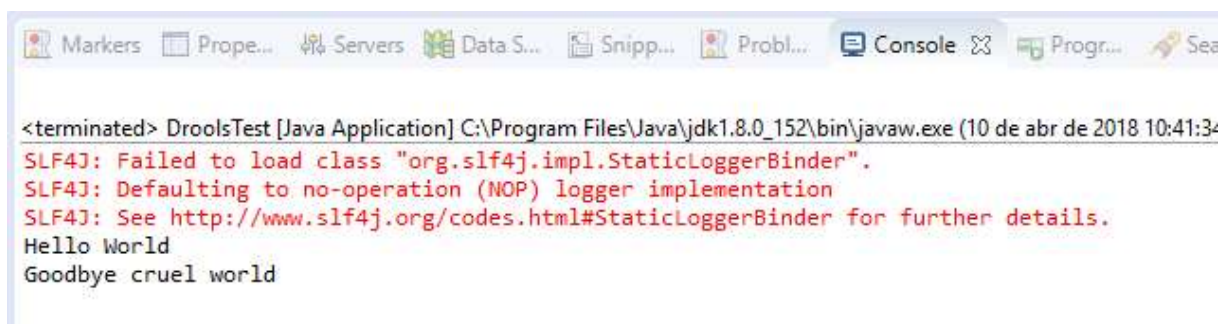
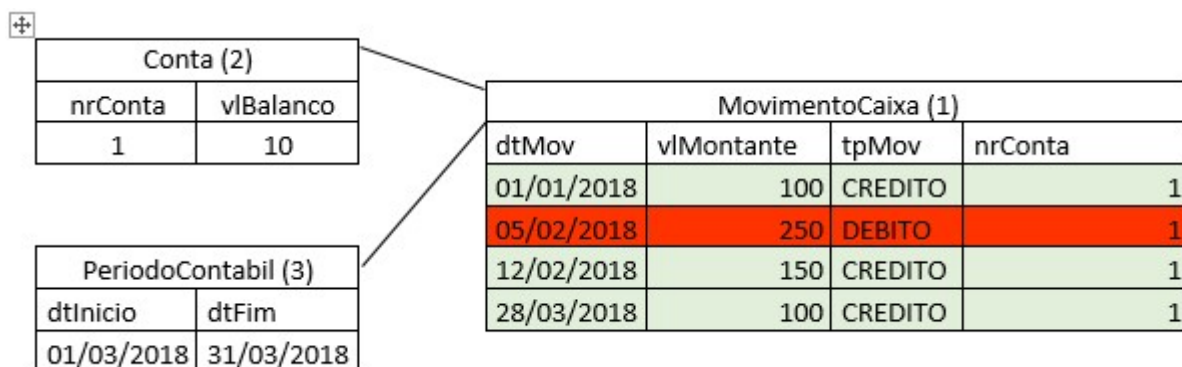


Figura 7: Mensagem de sucesso na execução

1.2. Modelo de dados utilizado no tutorial

O modelo abaixo foi retirado do livro citado no início (Héron), ele também pode ser encontrado na documentação oficial do drools (Drools), e descreve o seguinte cenário: Estamos em um banco que manipula contas (2) e em cada conta pode haver movimentos (1). O objetivo é calcular o saldo da conta entre um período contábil (3) de todas as contas.



□

Conta
- nrConta : long
- vlBalanco : double

PeriodoContabil
- dtInicio : Date
- dtFim : Date

MovimentacaoCaixa
- dtMov : Date
- vlMontante : double
- tpMov : int
- nrConta : long

1.3. Core: Projeto do modelo de dados

1. Crie um projeto vazio do Drools, chamado Banco, seguindo as mesmas instruções da criação do primeiro projeto, com a seguinte ressalva: Como é vazio, então você selecionará o primeiro botão da tela “*Select the Initial Project content*”. Pode “buildar” o projeto como Java e Runtime classes, apesar de não influenciar neste projeto, posteriormente você poderá convertê-lo para um projeto Maven, mas não se preocupe com isso agora.
2. Crie o pacote droolstutorial dentro de Banco/src/main/java;
3. Crie os POJOs de Conta, MovimentacaoConta e PeriodoContabil com as variáveis declaradas no modelo, seus getters(), setters() e toString(), dentro do pacote tutorialdrools.

- Conta.java

```
package droolstutorial;

public class Conta {
    private long nrConta;
    private double vlBalanco;

    public long getNrConta() {
        return nrConta;
    }
    public void setNrConta(long nrConta) {
        this.nrConta = nrConta;
    }
    public double getVlBalanco() {
        return vlBalanco;
    }
    public void setVlBalanco(double vlBalanco) {
        this.vlBalanco = vlBalanco;
    }

    @Override
    public String toString() {
        return "Conta [nrConta=" + nrConta + ", vlBalanco=" + vlBalanco +
        "];"
    }
}
```

- PeriodoContabil.java

```
package droolstutorial;

import java.util.Date;

public class PeriodoContabil {
    private Date dtlInicio;
    private Date dtFim;

    public Date getDtlInicio() {
        return dtlInicio;
    }
}
```

```

    }
    public void setDtInicio(Date dtInicio) {
        this.dtInicio = dtInicio;
    }
    public Date getDtFim() {
        return dtFim;
    }
    public void setDtFim(Date dtFim) {
        this.dtFim = dtFim;
    }

    @Override
    public String toString() {
        return "PeriodoContabil [dtInicio=" + dtInicio + ", dtFim=" + dtFim +
    "];
    }
}

```

- MovimentacaoCaixa.java

```

package droolstutorial;

import java.util.Date;

public class MovimentacaoCaixa {
    private Date dtMov;
    private double vlMontante;
    private int tpMov;
    private long nrConta;

    public Date getDtMov() {
        return dtMov;
    }
    public void setDtMov(Date dtMov) {
        this.dtMov = dtMov;
    }
    public double getVlMontante() {
        return vlMontante;
    }
    public void setVlMontante(double vlMontante) {
        this.vlMontante = vlMontante;
    }
    public int getTpMov() {
        return tpMov;
    }
    public void setTpMov(int tpMov) {
        this.tpMov = tpMov;
    }
    public long getNrConta() {
        return nrConta;
    }
    public void setNrConta(long nrConta) {
        this.nrConta = nrConta;
    }

    @Override
    public String toString() {
        return "MovimentacaoCaixa [dtMov=" + dtMov + ", vlMontante=" +
vlMontante + ", tpMov=" + tpMov + ", nrConta="

```



```

        + nrConta + "]]";
    }
}

```

4. Adicione a biblioteca do JUnit 4.x +.

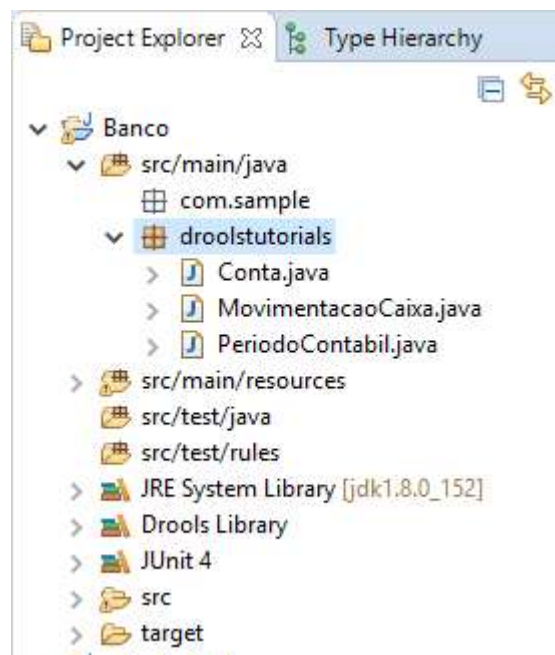
1. Selecione o projeto, clique com botão direito, selecione Build Path > Configure Build Path...
2. Na aba Libraries, clique em Add library... > JUnit > Selecione JUnit 4 > clique em Finish > Apply and Close.

Se você tiver convertido para um projeto Maven, então acrescente a dependência do JUnit no pom.xml.

5. Crie o Source Folder src/test/java e o src/test/rules.

1. File > New > Source Folder > ...

A estrutura do projeto deverá ficar assim:



6. Crie o pacote "util" n o Source Folder src/test/java.

7. Para facilitar o entendimento e escrita dos testes crie uma classe de ajuda chamada KnowledgeSessionHelper e a coloque no pacote util.

```
package util;
```

```

import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.StatelessKieSession;

```

```
public class KnowledgeSessionHelper {
```

```

public static KieContainer criarRegraBase() {

    KieServices ks = KieServices.Factory.get();
    KieContainer kieContainer = ks.getKieClasspathContainer();

    return kieContainer;
}

public static StatelessKieSession getStatelessKnowledgeSession(KieContainer kieContainer, String sessionName) {

    StatelessKieSession ksSession = kieContainer.newStatelessKieSession(sessionName);

    return ksSession;
}

public static KieSession getStateFullKnowledgeSession(KieContainer kieContainer, String sessionName) {
    KieSession kSession = kieContainer.newKieSession(sessionName);

    return kSession;
}
}

```

8. Crie o pacote droolstutorial no src/test/java e, dentro dele, crie um caso de teste com o nome PrimeiraTentativa;

1. Botão direito no src/test/java > New > Other > JUnit > JUnit Test Case ...

PrimeiraTentativa.java

```

package droolstutorial;

import org.junit.BeforeClass;
import org.junit.Test;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.StatelessKieSession;

import util.KnowledgeSessionHelper;

public class PrimeiraTentativa {

    StatelessKieSession sessionStateless = null;
    KieSession sessionStateFull = null;
    static KieContainer kieContainer;

    @BeforeClass
    public static void beforeClass() {
        kieContainer = KnowledgeSessionHelper.criarRegraBase();
    }

    @Test
    public void testeNumeroUm() {
        sessionStateFull = KnowledgeSessionHelper.getStateFullKnowledgeSession(kieContainer, "ksession-rules");
    }
}

```

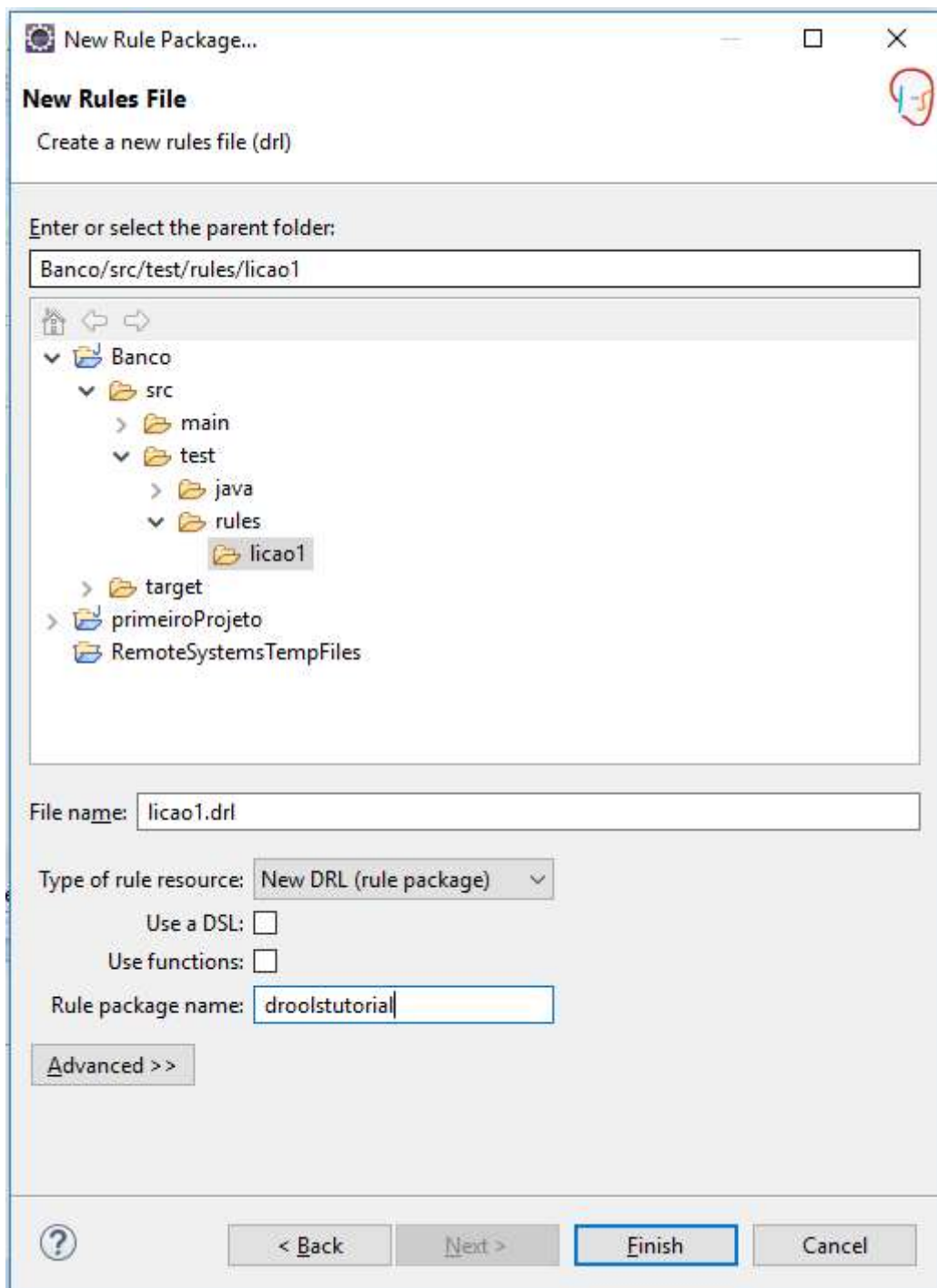
```
        sessionStateFull.fireAllRules();  
    }  
}
```

2. Execute o testeNumeroUm():

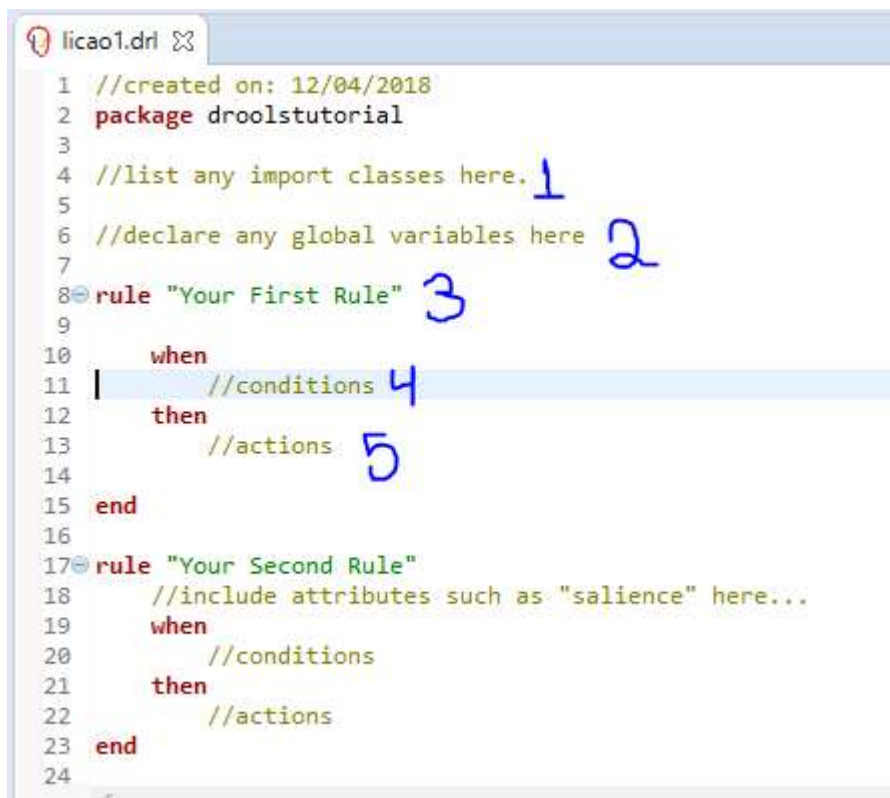
O teste deve passar para que possamos iniciar nossas lições.

1.3.1. Primeiro elemento da linguagem Drools

1. No projeto Banco, vá até src/test/rules e crie o pacote licao1:
2. Crie dentro deste pacote um arquivo de regras .drl (drools rule language):
 - a. File > New > Drools > Rule Resource > next.
File name: licao1;
Rule package name: droolstutorialç
Finish.



O seguinte elemento será exibido: licao1.drl.



```
1 //created on: 12/04/2018
2 package droolstutorial
3
4 //list any import classes here. 1
5
6 //declare any global variables here 2
7
8 rule "Your First Rule" 3
9
10   when
11   | //conditions 4
12   then
13   //actions 5
14
15 end
16
17 rule "Your Second Rule"
18   //include attributes such as "saliency" here...
19   when
20   //conditions
21   then
22   //actions
23 end
24
```

Analisando o arquivo, temos:

1. Para cada objeto java que for utilizar, você precisará importar uma classe;
2. É possível definir variáveis globais;
3. O nome da regra é único;
4. Parte condicional da regra, algumas vezes são chamadas de LHS (Left Hand Side), que segue uma sintaxe que será abordada logo mais;
5. O RHS (Right Hand Side) é basicamente um bloco que permite que o código semântico específico do dialeto seja executado. Aqui é possível escrever um código java puro.

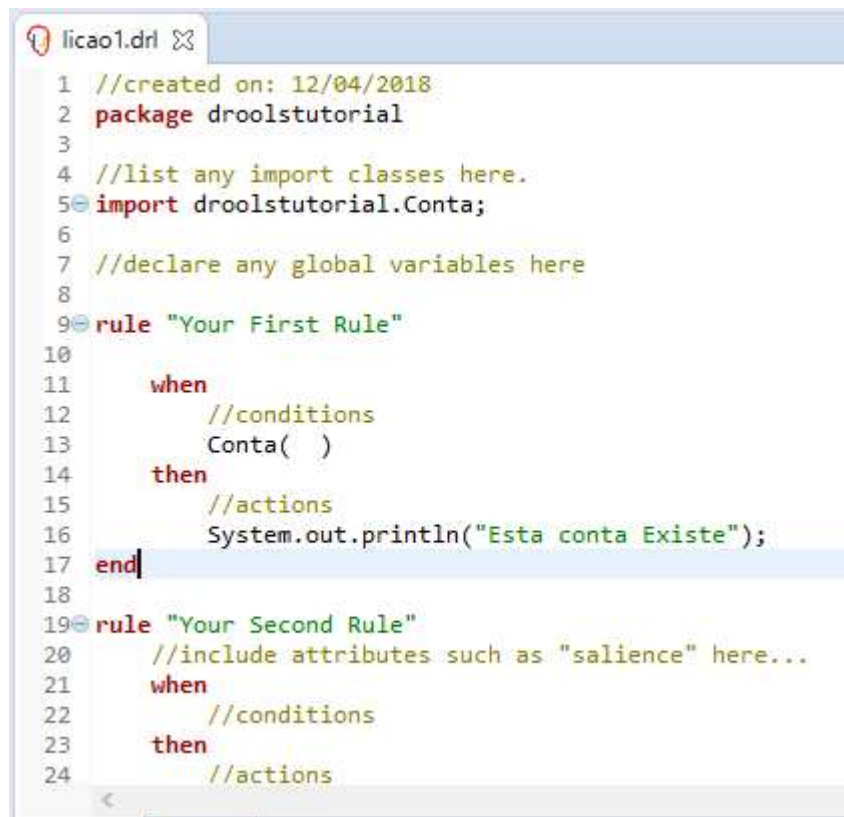
1.3.2. Adicionando uma condição simples

Como qualquer plugin do eclipse, o plugin do drools auto completa utilizando o ctrl+space.

Crie uma regra que tem um fato do tipo “Conta”. Se a regra for executada, então será exibido no console a mensagem “Esta conta existe”.

1. Importe a classe Conta;

2. No “when” acrescente o fato Conta()
3. No “then” acrescente o sysout com a mensagem.



```

1 //created on: 12/04/2018
2 package droolstutorial
3
4 //list any import classes here.
5 import droolstutorial.Conta;
6
7 //declare any global variables here
8
9 rule "Your First Rule"
10
11     when
12         //conditions
13         Conta( )
14     then
15         //actions
16         System.out.println("Esta conta Existe");
17 end
18
19 rule "Your Second Rule"
20     //include attributes such as "salience" here...
21     when
22         //conditions
23     then
24         //actions

```

4. Crie um caso de teste do JUnit em src/test/java/droolstutorial, chamado TesteLicao1 e modifique-o para ficar assim:

```

package droolstutorial;

import org.junit.BeforeClass;
import org.junit.Test;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.StatelessKieSession;

import util.KnowledgeSessionHelper;

public class TesteLicao1 {
    StatelessKieSession sessionStateless = null;
    KieSession sessionStateFull = null;
    static KieContainer kieContainer;

    @BeforeClass
    public static void beforeClass() {
        kieContainer = KnowledgeSessionHelper.criarRegraBase();
    }

    @Test
    public void testeNumeroUm() {
        sessionStateFull = KnowledgeSessionHelper.getStateFullKnowledgeSession(kieContainer, "ksession-rules");
    }

```

```
//      Fato Conta
Conta conta = new Conta();
sessionStateFull.insert(conta);

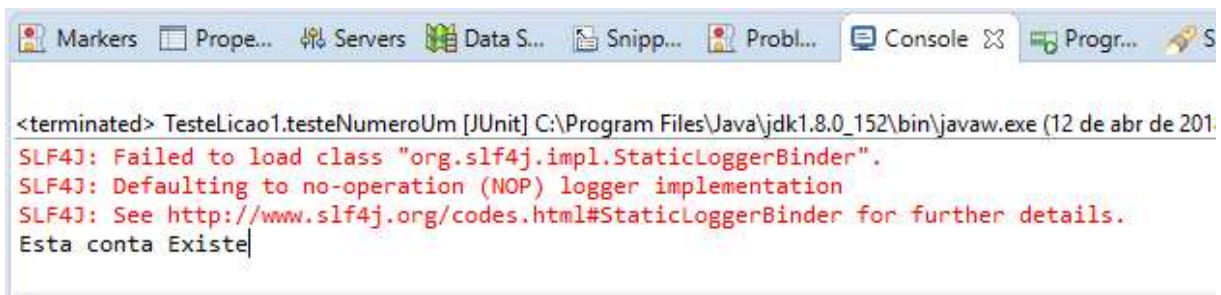
sessionStateFull.fireAllRules();
    }
}
```

5. Em src/main/resource/META-INF, modifique o kmodule.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns="http://jboss.org/kie/6.0.0/kmodule">
  <kbase name="rules" packages="licao1">
    <ksession name="ksession-rules">
  </kbase>
</kmodule>
```

Acima foi definida a sessão chamada "*ksession-rules*", e o pacote as regras foram definidas, o *packages="licao1"*.

6. Execute o teste da classe TesteLicao1, que deve aparecer isto:



```
<terminated> TesteLicao1.testeNumeroUm [JUnit] C:\Program Files\Java\jdk1.8.0_152\bin\javaw.exe (12 de abr de 201
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.
Esta conta Existe|
```

1.3.3. Usando variáveis globais

Com global você define variáveis globais. Eles são usados para disponibilizar objetos de aplicativo para as regras. Normalmente, eles são usados para fornecer dados ou serviços que as regras usam, especialmente serviços de aplicativos usados em consequências de regras, e para retornar dados das regras, como logs ou valores adicionados nas consequências da regra, ou para as regras interagirem com o aplicativo, fazendo callbacks. Globais não são inseridos na Memória de Trabalho e, portanto, um global nunca deve ser usado para estabelecer condições em regras, exceto quando tiver um valor imutável constante. O mecanismo não pode ser notificado sobre alterações de valor de globais e não rastreia suas alterações. O uso incorreto de globais em restrições pode produzir resultados surpreendentes - surpreendendo de uma maneira ruim.

Se vários pacotes declararem globais com o mesmo identificador, eles devem ser do mesmo tipo e todos eles referenciarão o mesmo valor global.

No código que acabamos de fazer, escrevemos o código `System.out.println ("bla-bla")`. Isso é bom, mas imagine que você deseja o log em outro lugar, não é possível.

Uma boa prática é usar variáveis globais para esse propósito.

1. Primeiro defina uma classe java chamada `OutputDisplay` no pacote `util` em `src/main/java`. Ficando Assim:

```
package util;

public class OutputDisplay {

    public void exibeTexto(String algumTexto) {
        long hora = System.currentTimeMillis();

        System.out.println("hora = " + hora + " - " + algumTexto);
    }
}
```

2. Atualize o arquivo de regras `licao1.drl`, deixando assim:

```
//created on: 12/04/2018
package droolstutorial

//list any import classes here.
import droolstutorial.Conta;
import util.OutputDisplay;

//declare any global variables here
global OutputDisplay resultado;

rule "Your First Rule revisada"

    when
        //conditions
        Conta( )
    then
        //actions
        resultado.exibeTexto("Esta conta Existe");
    end
```

A palavra-chave “global” é usada e, em seguida, uma declaração normal do java e acrescentada. Aqui o global é do tipo `OutputDisplay` e a variável é chamada de `resultado`. Essa variável agora pode ser usada na parte RHS da regra.

3. Para inicializar a variável global, use o método `setGlobal` na sessão que criada em `TesteLicao1`, conforme mostrado aqui:

```
@Test
public void testeNumeroUm() {
    sessionStateFull = KnowledgeSessionHelper.getStateFullKnowledgeSession(kieContainer, "ksession-rules");
}
```

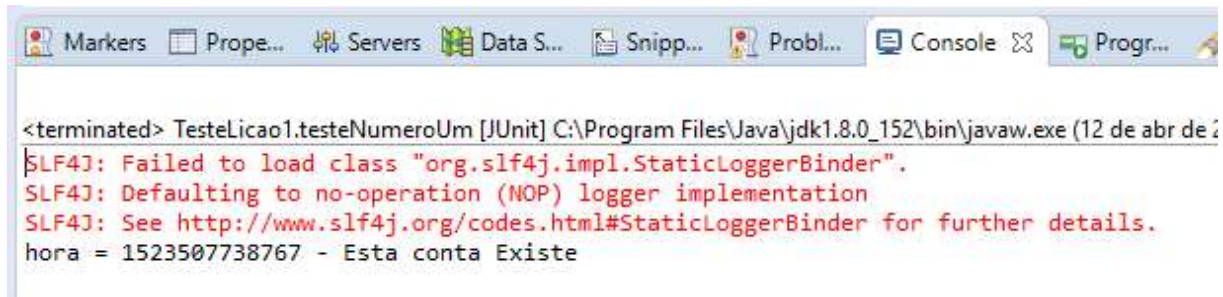


```
//          Variavel global
OutputDisplay outputDisplay = new OutputDisplay();
sessionStateFull.setGlobal("resultado", outputDisplay);

//          Fato Conta
Conta conta = new Conta();
sessionStateFull.insert(conta);

sessionStateFull.fireAllRules();
}
```

4. Execute o teste, no console deve aparecer algo como isto:



```
<terminated> TesteLicao1.testeNumeroUm [JUnit] C:\Program Files\Java\jdk1.8.0_152\bin\javaw.exe (12 de abr de 2017)
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.
hora = 1523507738767 - Esta conta Existe
```

1.3.4. Usando call-backs para registrar as atividades no Drools Runtime

Até agora só definimos uma regra. Ela é executada ou não e, quando executada, adicionamos um método que nos mostra uma mensagem.

Em projetos maiores, adicionar código de log a cada regra não é uma boa prática e irá aumentar a complexidade da escrita delas e, além disso, forçaria o analista de negócios a escrever um código técnico.

O Drools oferece um padrão para implementar essa funcionalidade que é chamada de callbacks de sessão:

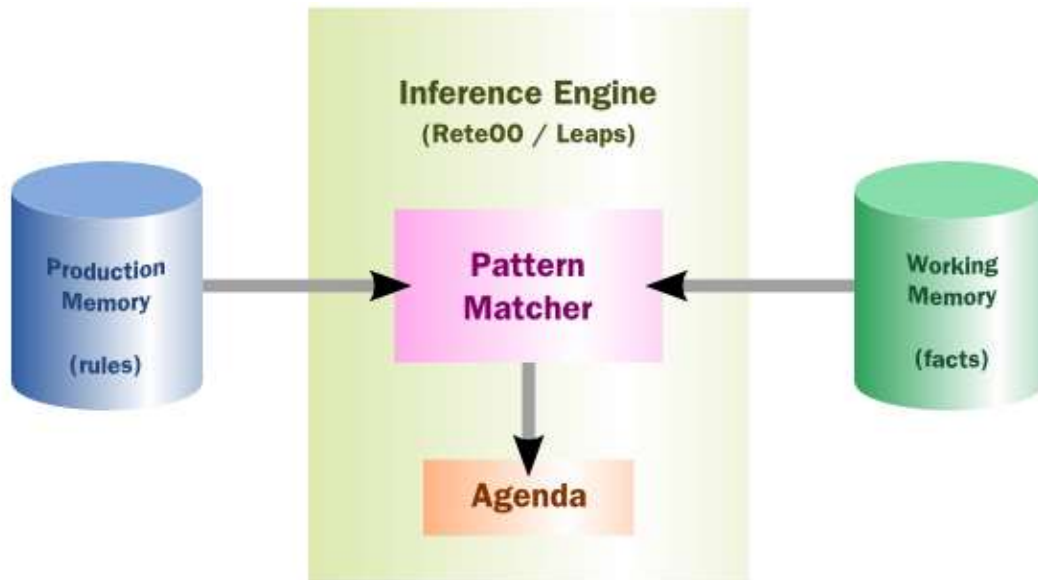


Figura 8: Visão de alto nível de um sistema de regras de produção

1. A Production Memory contém toda a definição de regra (no nosso caso o drl para o momento);
2. Os fatos inseridos no método insert são adicionados a Working Memory criada na sessão;
3. A Agenda contém todas as regras que podem ser disparadas.
4. O Pattern Matcher é o algoritmo usado para corresponder às regras dos fatos fornecidos. Na versão mais recente do drools, existem muitos algoritmos diferentes que são usados (O principal deles é o algoritmo RETE).

Em cada uma dessas partes, é possível adicionar um retorno de chamada quando criamos uma sessão:

```
sessionStateFull.addListener(new RuleRuntimeEventListener() {

    @Override
    public void objectUpdated(ObjectUpdatedEvent arg0) {
        System.out.println("Objeto atualizado: \n"
            + arg0.getObject().toString());
    }

    @Override
    public void objectInserted(ObjectInsertedEvent arg0) {
        System.out.println("Objeto inserido: \n"
            + arg0.getObject().toString());
    }

    @Override
    public void objectDeleted(ObjectDeletedEvent arg0) {
        System.out.println("Objeto removido: \n"
            + arg0.getObject().toString());
    }
});
```

```

        + arg0.getOldObject().toString());
    }
});

```

Toda vez que um fato é inserido, atualizado ou removido o argumento do log é exibido no console. Ele usa o método toString() da instancia do java passada – do fato – (arg0.getObject().toString()).

Crie um novo de teste unitário na classe TesteLicao1, acrescentando o retorno do log da sessão e execute todos os testes:

```

@Test
public void testeComFatoEUsandoCallBackGlobal() {
    sessionStateFull = KnowledgeSessionHelper.getStateFullKnowledgeSession(kieContainer,
    "ksession-rules");

    sessionStateFull.addEventListener(new RuleRuntimeEventListener() {

        @Override
        public void objectUpdated(ObjectUpdatedEvent arg0) {
            System.out.println("Objeto atualizado: \n"
                + arg0.getObject().toString());
        }

        @Override
        public void objectInserted(ObjectInsertedEvent arg0) {
            System.out.println("Objeto inserido: \n"
                + arg0.getObject().toString());
        }

        @Override
        public void objectDeleted(ObjectDeletedEvent arg0) {
            System.out.println("Objeto removido: \n"
                + arg0.getOldObject().toString());
        }
    });

    // Fato Conta
    Conta conta = new Conta();
    conta.setNrConta(1);

    FactHandle factHandle = sessionStateFull.insert(conta);
    conta.setVIBalanco(12.0);

    sessionStateFull.update(factHandle, conta);
    sessionStateFull.delete(factHandle);
    sessionStateFull.fireAllRules();

    System.out.println("\nPercebeu?");
}

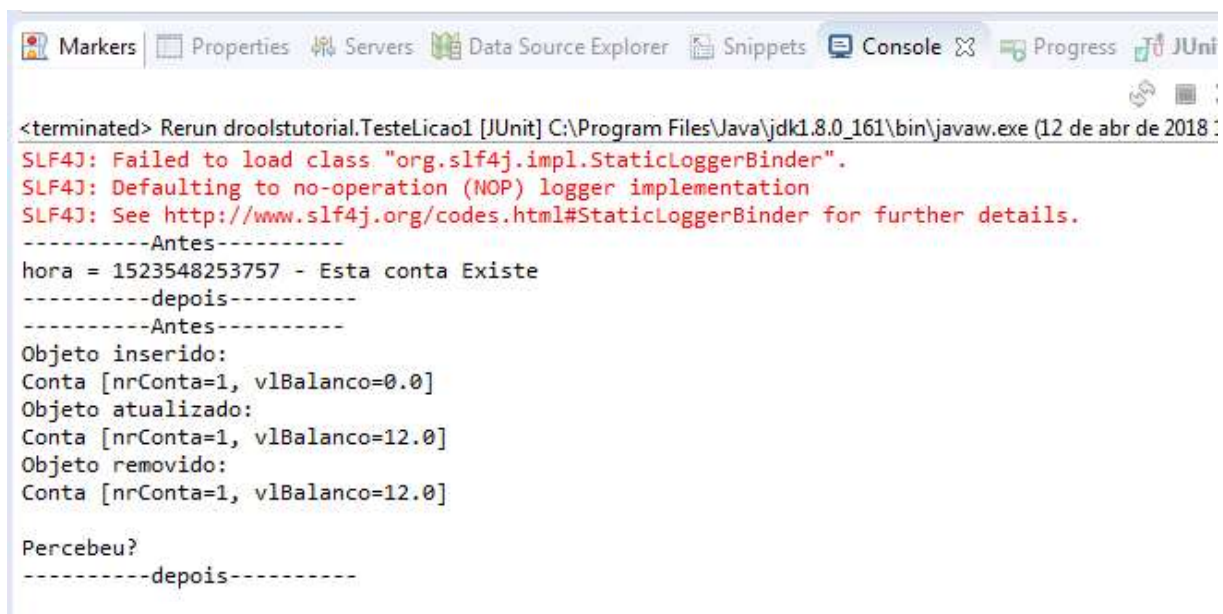
```

A primeira linha é a execução do primeiro teste. Para diferenciar melhor quando um novo teste é iniciado, acrescente o código abaixo no JUnit.

```
@Before
public void antes() {
    System.out.println("-----Antes-----");
}

@After
public void depois() {
    System.out.println("-----depois-----");
}
```

Executando novamente, aparecerá assim:



```
<terminated> Rerun droolstutorial.TesteLicao1 [JUnit] C:\Program Files\Java\jdk1.8.0_161\bin\javaw.exe (12 de abr de 2018 :
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.
-----Antes-----
hora = 1523548253757 - Esta conta Existe
-----depois-----
-----Antes-----
Objeto inserido:
Conta [nrConta=1, vlBalanco=0.0]
Objeto atualizado:
Conta [nrConta=1, vlBalanco=12.0]
Objeto removido:
Conta [nrConta=1, vlBalanco=12.0]

Percebeu?
-----depois-----
```

O primeiro teste é executado e é possível ver o output gerado quando a regra é executada. Já o segundo teste, quando executado, primeiro tem um objeto inserido, depois é atualizado e removido – você conseguirá visualizar melhor o comportamento se debugar cada evento de log.

Para atualizar um objeto, primeiro você memoriza um Fact Handle:

```
FactHandle factHandle = sessionStateFull.insert(conta);
```

Ao atualizar este factHandle, você então avisa para o Drools que o objeto foi atualizado:

```
conta.setVlBalanco(12.0);
sessionStateFull.update(factHandle, conta);
```

Então você pede para ele deletar:

```
sessionStateFull.update(factHandle, conta);
sessionStateFull.delete(factHandle);
```

Como chamou fireAllRules() depois que retirou o único fato que estava na memória de trabalho, a regra “Your First Rule revisada” não é disparada.

1.3.5. Quando e como uma regra é executada

Faça o seguinte exemplo, ainda em TesteLicao1.java:

```
@Test
public void testeComFatoEDoisFireAllRules() {
    sessionStateFull = KnowledgeSessionHelper.getStateFullKnowledgeSession(kieContainer,
    "ksession-rules");

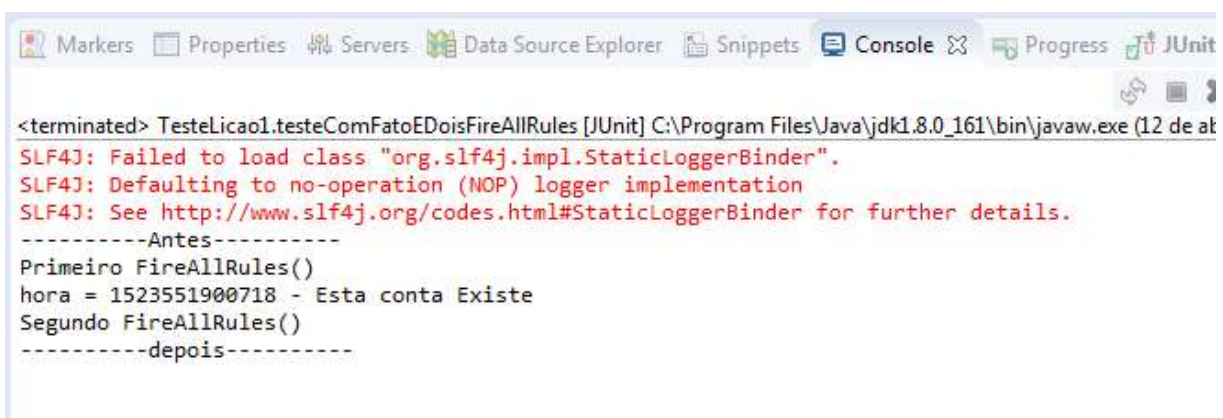
    // Variavel global
    OutputDisplay outputDisplay = new OutputDisplay();
    sessionStateFull.setGlobal("resultado", outputDisplay);

    // Fato Conta
    Conta conta = new Conta();
    sessionStateFull.insert(conta);

    System.out.println("Primeiro FireAllRules()");
    sessionStateFull.fireAllRules();
    System.out.println("Segundo FireAllRules()");
    sessionStateFull.fireAllRules();
}
```

Você inseriu uma conta, mandou executar a regra e depois mandou executá-la novamente. Se pergunte: O que irá acontecer ao executar este teste? Quantas vezes a regra “Your First Rule revisada” será executada?

Veja o resultado:



```
<terminated> TesteLicao1.testeComFatoEDoisFireAllRules [JUnit] C:\Program Files\Java\jdk1.8.0_161\bin\javaw.exe (12 de at
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.
-----Antes-----
Primeiro FireAllRules()
hora = 1523551900718 - Esta conta Existe
Segundo FireAllRules()
-----depois-----
```

Resposta: A regra é executada uma única vez.

Talvez isso aconteceu porque a Conta não foi modificada?

Faça outro exemplo modificando o objeto Conta que está sendo passado na sessão, assim:

```
@Test
public void testeComFatoEDoisFireAllRulesEUmSetter() {
    sessionStateFull = KnowledgeSessionHelper.getStateFullKnowledgeSession(kieContainer,
    "ksession-rules");

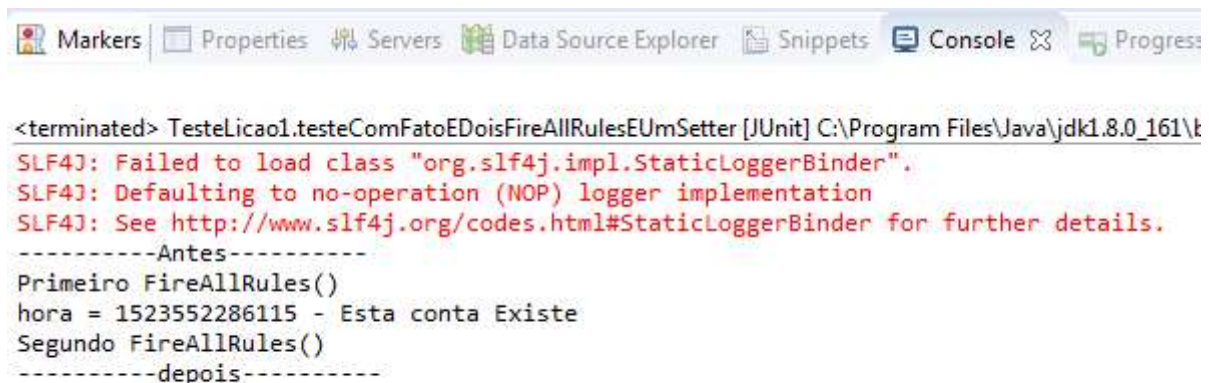
    // Variavel global
    OutputDisplay outputDisplay = new OutputDisplay();
    sessionStateFull.setGlobal("resultado", outputDisplay);

    // Fato Conta
    Conta conta = new Conta();
    sessionStateFull.insert(conta);

    System.out.println("Primeiro FireAllRules()");
    sessionStateFull.fireAllRules();
    conta.setNrConta(1);

    System.out.println("Segundo FireAllRules()");
    sessionStateFull.fireAllRules();
}
```

Veja o resultado:



```
<terminated> TesteLicao1.testeComFatoEDoisFireAllRulesEUmSetter [JUnit] C:\Program Files\Java\jdk1.8.0_161\
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.
-----Antes-----
Primeiro FireAllRules()
hora = 1523552286115 - Esta conta Existe
Segundo FireAllRules()
-----depois-----
```

A regra continua do mesmo jeito, sem ser executada novamente, é preciso informar para o Rule Engine que o fato foi modificado. Faça outro teste assim:

```
@Test
public void testeComFatoDoisFireAllRulesUmSetterEUpdate() {
    sessionStateFull = KnowledgeSessionHelper.getStateFullKnowledgeSession(kieContainer,
    "ksession-rules");

    // Variavel global
    OutputDisplay outputDisplay = new OutputDisplay();
    sessionStateFull.setGlobal("resultado", outputDisplay);

    // Fato Conta
    Conta conta = new Conta();
    FactHandle factHandle = sessionStateFull.insert(conta);
```

```

        System.out.println("Primeiro FireAllRules()");
        sessionStateFull.fireAllRules();

        conta.setNrConta(1);

        sessionStateFull.update(factHandle, conta);
        System.out.println("Segundo FireAllRules()");
        sessionStateFull.fireAllRules();
    }

```

A regra foi executada pela segunda vez.

Entenda o que acontece quando o fireAllRules() é chamado numa sessão Statefull:

1. O Drools vai procurar todas as regras que podem ser aplicadas e as coloca na Agenda;
2. Então ele executa a regra que está no topo da Agenda;
3. Depois de executada, a regra é desativada;
4. Você precisa informar para o drools que o estado do fato mudou na parte “when” (LHS) fazendo-o reconsiderar a regra;
5. Este estado pode ser um insert, update ou delete (retract);

No exemplo anterior, foi dito para o Drools que o fato foi atualizado:

```
sessionStateFull.update(factHandle, conta);
```

Sendo assim, considerando que o fato inserido foi atualizado, o Drools reexecuta a regra.

Como na regra “Your First Rule revisada” não há atributos na condição, a regra é executada passando apenas o fato Conta(). Porém o mesmo pode ser feito na parte do “then” (RHS): insert, update e delete.

Veja este outro teste:

```

@Test
public void testeComFatoQueInsereObjeto() {
    sessionStateFull = KnowledgeSessionHelper.getStateFullKnowledgeSession(kieContainer,
    "ksession-rules");

    // Variavel global
    OutputDisplay outputDisplay = new OutputDisplay();
    sessionStateFull.setGlobal("resultado", outputDisplay);

    sessionStateFull.addEventListener(new RuleRuntimeEventListener() {
        @Override
        public void objectUpdated(ObjectUpdatedEvent arg0) {
            System.out.println("Objeto atualizado: \n"
                + arg0.getObject().toString());
        }
        @Override
        public void objectInserted(ObjectInsertedEvent arg0) {
            System.out.println("Objeto inserido: \n"
                + arg0.getObject().toString());
        }
    });
}

```



```

    }
    @Override
    public void objectDeleted(ObjectDeletedEvent arg0) {
        System.out.println("Objeto removido: \n"
            + arg0.getOldObject().toString());
    }
});

// Fato Movimentação do Caixa
MovimentacaoCaixa movimentacaoCaixa = new MovimentacaoCaixa();
FactHandle factHandle = sessionStateFull.insert(movimentacaoCaixa);

sessionStateFull.fireAllRules();
}

```

Acrescente as regras abaixo no `licao1.drl`:

```

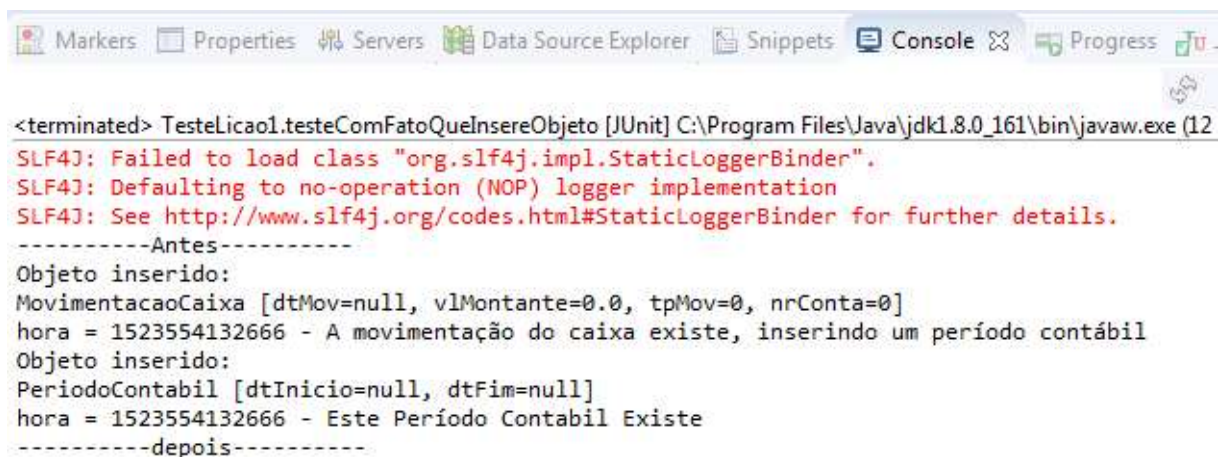
rule "Your First Rule revisada MovimentacaoCaixa"

    when
        MovimentacaoCaixa( )
    then
        resultado.exibeTexto("A movimentação do caixa existe, inserindo um período contábil");
        PeriodoContabil novoPeriodoContabil = new PeriodoContabil();
        insert(novoPeriodoContabil);
    end

rule "Período Contabil inserida"
    when
        PeriodoContabil( )
    then
        resultado.exibeTexto("Este Período Contabil Existe");
    end

```

Execute o teste, você terá um resultado assim:



```

<terminated> TesteLicao1.testeComFatoQueInseriObjeto [JUnit] C:\Program Files\Java\jdk1.8.0_161\bin\javaw.exe (12
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.
-----Antes-----
Objeto inserido:
MovimentacaoCaixa [dtMov=null, vlMontante=0.0, tpMov=0, nrConta=0]
hora = 1523554132666 - A movimentação do caixa existe, inserindo um período contábil
Objeto inserido:
PeriodoContabil [dtInicio=null, dtFim=null]
hora = 1523554132666 - Este Período Contabil Existe
-----depois-----

```

Analisando o log, você verá o seguinte:

1. Duas primeiras linhas: Um objeto do tipo `MovimentacaoCaixa` foi inserido, isso foi feito pelo código no teste do JUnit.


```
FactHandle factHandle = sessionStateFull.insert(movimentacaoCaixa);
```

2. Terceira linha: A mensagem exibida foi gerada na parte do “then” da "Your First Rule revisada MovimentacaoCaixa";
3. Quarta e quinta linha: Um objeto do tipo PeriodoContabil foi inserido, isso foi feito na parte “then” do "Your First Rule revisada MovimentacaoCaixa":

```
PeriodoContabil novoPeriodoContabil = new PeriodoContabil();  
insert(novoPeriodoContabil);
```
4. A última linha é o resultado da execução da regra "Periodo Contabil inserida".

1.4. Binding: Relacionando fatos e atributos em uma regra

Para que você possa ver o que está acontecendo na Rule Engine, adicione o método abaixo no KnowledgeSessionHelper.

```
public static KieSession getStatefulKnowledgeSessionWithCallback( KieContainer kieContainer, String sessionName) {  
    KieSession session = getStatefulKnowledgeSession(kieContainer, sessionName);  
    session.addListener(new RuleRuntimeEventListener() {  
        public void objectInserted(ObjectInsertedEvent event) {  
            System.out.println("Objecto inserido \n"  
                + event.getObject().toString());  
        }  
        public void objectUpdated(ObjectUpdatedEvent event) {  
            System.out.println("Objecto foi atualizado \n"  
                + "new Content \n" + event.getObject().toString());  
        }  
        public void objectDeleted(ObjectDeletedEvent event) {  
            System.out.println("Objecto removido \n"  
                + event.getOldObject().toString());  
        }  
    });  
    session.addListener(new AgendaEventListener() {  
        public void matchCreated(MatchCreatedEvent event) {  
            System.out.println("A regra \"  
                + event.getMatch().getRule().getName()  
                + \"' pode ser executada na agenda");  
        }  
        public void matchCancelled(MatchCancelledEvent event) {  
            System.out.println("A regra \"  
                + event.getMatch().getRule().getName()  
                + \"' não pode ser executada na agenda");  
        }  
        public void beforeMatchFired(BeforeMatchFiredEvent event) {  
            System.out.println("A regra \"  
                + event.getMatch().getRule().getName()  
                + \"' será executada");  
        }  
        public void afterMatchFired(AfterMatchFiredEvent event) {  
            System.out.println("A regra \"  
                + event.getMatch().getRule().getName()  
                + \"' foi executada");  
        }  
        public void agendaGroupPopped(AgendaGroupPoppedEvent event) {
```

```

    }
    public void agendaGroupPushed(AgendaGroupPushedEvent event) {
    }
    public void beforeRuleFlowGroupActivated(RuleFlowGroupActivatedEvent event) {
    }
    public void afterRuleFlowGroupActivated(RuleFlowGroupActivatedEvent event) {
    }
    public void beforeRuleFlowGroupDeactivated(RuleFlowGroupDeactivatedEvent event) {
    }
    public void afterRuleFlowGroupDeactivated(RuleFlowGroupDeactivatedEvent event) {
    }
  });

  return session;
}

```

Acrescente no pacote “util” a classe DateHelper abaixo:

```

package util;

import java.text.SimpleDateFormat;
import java.util.Date;

public class DateHelper {
    public static String sFormat = "yyyy-MM-dd";

    public static Date getDate(String sDate) throws Exception {
        SimpleDateFormat sdf = new SimpleDateFormat(sFormat);
        return sdf.parse(sDate);
    }

    public static Date getDate(String sDate, String anotherFormat)
        throws Exception {
        SimpleDateFormat sdf = new SimpleDateFormat(anotherFormat);
        return sdf.parse(sDate);
    }
}

```

Deixe o kmodule.xml assim:

```

<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns="http://jboss.org/kie/6.0.0/kmodule">
  <kbase name="rules" packages="licao1">
    <ksession name="ksession-rules"/>
  </kbase>
  <kbase name="rules2" packages="licao2">
    <ksession name="ksession-licao2"/>
  </kbase>
</kmodule>

```

Em src/test/rules crie o pacote licao2 e a regra licao2.drl, que ficará assim:

```

//created on: 12/04/2018
package droolstutorial

//list any import classes here.
import droolstutorial.Conta;

```

```

import droolstutorial.MovimentacaoCaixa;
import droolstutorial.PeriodoContabil;

import util.OutputDisplay;

//declare any global variables here
global OutputDisplay resultado;

rule "Your First Rule revisited again"

    when
        Conta( )
    then
        resultado.exibeTexto("Esta conta Existe");
    end

```

Na classe MovimentacaoCaixa, acrescente as seguintes alterações:

```

public static int CREDITO = 1;
public static int DEBITO = 2;

@Override
public String toString() {
    // TODO Auto-generated method stub
    StringBuffer buff = new StringBuffer();
    buff.append("-----Movimentação do Caixa-----\n");
    buff.append("Número da conta=" + this.nrConta + "\n");
    if (this.dtMov != null) {
        buff.append("Data do movimento="
            + DateFormat.getDateInstance().format(this.dtMov)
            + "\n");
    } else {
        buff.append("Nenhuma data de movimento setada\n");
    }
    buff.append("Montante movimentado=" + this.vlMontante + "\n");
    buff.append("-----Movimentação do Caixa-----");
    return buff.toString();
}

```

Crie outra classe de teste chamada TesteLicao2.java:

```

package droolstutorial;

import org.junit.After;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.StatelessKieSession;

import util.KnowledgeSessionHelper;
import util.OutputDisplay;

public class TesteLicao2 {

```

```

static KieContainer kieContainer;
StatelessKieSession sessionStateless = null;
KieSession sessionStatefull = null;

@BeforeClass
public static void beforeClass() {
    kieContainer = KnowledgeSessionHelper.criarRegraBase();
}

@Before
public void antes() {
    System.out.println("-----Antes-----");
}

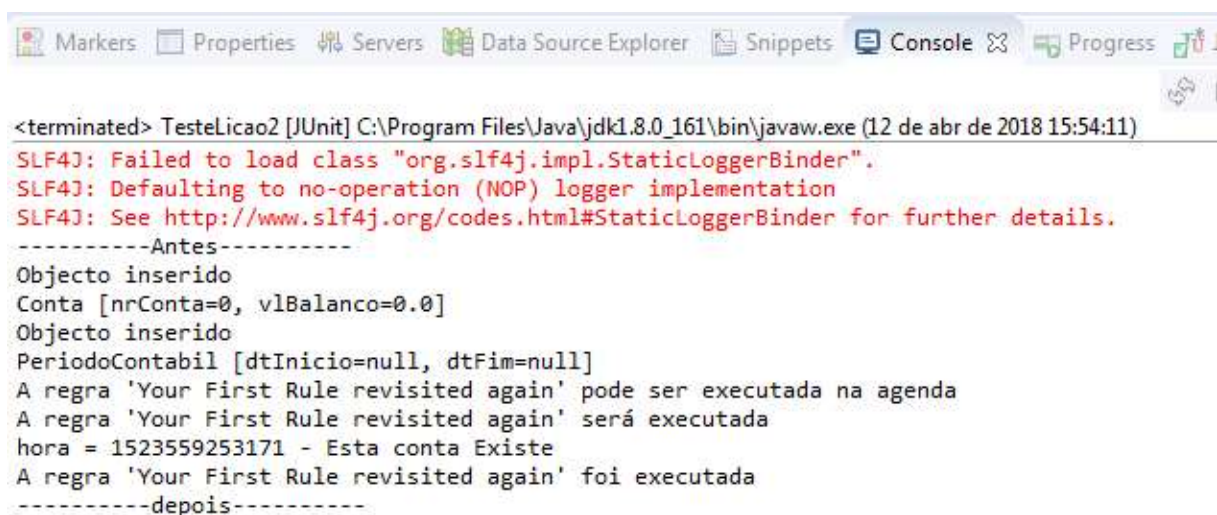
@After
public void depois() {
    System.out.println("-----depois-----");
}

@Test
public void test2Fatos() {
    sessionStatefull = KnowledgeSessionHelper
        .getStatefulKnowledgeSessionWithCallback(kieContainer,"ksession-licao2");

    OutputDisplay display = new OutputDisplay();
    sessionStatefull.setGlobal("resultado", display);
    Conta conta = new Conta();
    sessionStatefull.insert(conta);
    PeriodoContabil periodoContabil = new PeriodoContabil();
    sessionStatefull.insert(periodoContabil);
    sessionStatefull.fireAllRules();
}
}

```

Se ao executar o teste aparecer algo como a imagem abaixo, então poderá continuar com as próximas regras:



```

<terminated> TesteLicao2 [JUnit] C:\Program Files\Java\jdk1.8.0_161\bin\javaw.exe (12 de abr de 2018 15:54:11)
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.
-----Antes-----
Objecto inserido
Conta [nrConta=0, vlBalanco=0.0]
Objecto inserido
PeriodoContabil [dtInicio=null, dtFim=null]
A regra 'Your First Rule revisited again' pode ser executada na agenda
A regra 'Your First Rule revisited again' será executada
hora = 1523559253171 - Esta conta Existe
A regra 'Your First Rule revisited again' foi executada
-----depois-----

```

Comente o trecho de código da regra acima, pois não a utilizaremos por agora.

1.4.1. Caso de Teste

Nesta segunda lição você irá implementar um caso de teste com os seguintes dados: Conta: número 1, um Período Contábil que vai de primeiro de janeiro de 2016 à 31 de março de 2016, uma Movimentação de Caixa com crédito de R\$ 1.000,00 em 15 de janeiro de 2016, com débito de R\$ 500,00 em fevereiro de 2016 e um crédito de R\$ 1.000,00 em 15 de abril de 2016.

O resultado deverá ser um balanço de R\$ 500,00 para o período contábil apresentado.

1.4.2. Ligação dos Fatos

Agora, em uma nova regra, você irá atualizar o balanço contábil de cada movimentação do caixa sempre que houver um lançamento de crédito. Primeiro coloque o fato `MovimentacaoCaixa` e, dentro dele, vincule o tipo do movimento do fato com o tipo do movimento `CRÉDITO` da classe Java `MovimentacaoCaixa`. Em seguida, acrescente o fato `Conta`. Veja abaixo como deverá ficar.

```
rule "Regra Crédito"
when
    MovimentacaoCaixa( tpMov == MovimentacaoCaixa.CREDITO )
    Conta( )
then
    ??
end
```

Na frente dos fatos você pode declarar variáveis que podem ser usadas para parte do “then” como mostrado abaixo (o \$ só é utilizado para ajudar a distinguir o que é java do que é drools, não é obrigatório no código).

```
rule "Regra Crédito"
when
    $movCa: MovimentacaoCaixa( tpMov == MovimentacaoCaixa.CREDITO )
    $conta: Conta( )
then
    $conta.setVIBalanco($conta.getVIBalanco() + $movCa.getVIMontante());
    resultado.exibeTexto("Conta número: " + $conta.getNrConta() + " tem agora um balanço de
R$ " + $conta.getVIBalanco());
end
```

Na classe Java, crie outro teste que passe dois fatos como no exemplo anterior, porém passando valores para a conta e para a movimentação, como no exemplo abaixo:

```
@Test
public void test2FatosPopulados() {
    sessionStatefull = KnowledgeSessionHelper
        .getStatefulKnowledgeSessionWithCallback(kieContainer,"ksession-licao2");
}
```

```

OutputDisplay display = new OutputDisplay();
sessionStatefull.setGlobal("resultado", display);

Conta conta = new Conta();
conta.setNrConta(1);
conta.setVIBalanco(0);
sessionStatefull.insert(conta);

MovimentacaoCaixa caixa = new MovimentacaoCaixa();
caixa.setNrConta(1);
caixa.setTpMov(MovimentacaoCaixa.CREDITO);
caixa.setVIMontante(1000.0);
sessionStatefull.insert(caixa);

sessionStatefull.fireAllRules();

Assert.assertEquals(1000.0, conta.getVIBalanco(), 0);
}

```

Execute o teste e obtenha algo similar a imagem abaixo:

```

<terminated> Rerun droooltutorial.TesteLicao2.test2FatosPopulados [JUnit] C:\Program Files\Java\jdk1.8.0_1
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
-----Antes-----
Objecto inserido
Conta [nrConta=1, vlBalanco=0.0]
Objecto inserido
-----Movimentação do Caixa-----)
Número da conta=1
Nenhuma data de movimento setada
Montante movimentado=1000.0
-----Movimentação do Caixa-----)
A regra 'Regra Crédito' pode ser executada na agenda
A regra 'Regra Crédito' será executada
hora = 1523833000431 - Conta número: 1 tem agora um balanço de R$ 1000.0
A regra 'Regra Crédito' foi executada
-----depois-----

```

Resumindo, toda vez que você passar os fatos Conta e MovimentacaoCaixa, este último com o tipo Crédito atribuído a ele, então a “Regra Crédito” será executada.

1.4.3. Ligação de Atributos

Faça agora outro teste com um segundo movimento de caixa atribuindo o mesmo valor do montante, porém atribuindo para outro número de conta, como mostrado abaixo:

```

@Test
public void test2FatosPopuladosComContasDiferentes() throws Exception {
    sessionStatefull = KnowledgeSessionHelper

```

```
.getStatefulKnowledgeSessionWithCallback(kieContainer,"ksession-licao2");
```

```
OutputDisplay display = new OutputDisplay();  
sessionStatefull.setGlobal("resultado", display);
```

```
Conta conta = new Conta();  
conta.setNrConta(1);  
conta.setVIBalanco(0);  
sessionStatefull.insert(conta);
```

```
MovimentacaoCaixa caixa = new MovimentacaoCaixa();  
caixa.setNrConta(1);  
caixa.setTpMov(MovimentacaoCaixa.CREDITO);  
caixa.setVIMontante(1000.0);  
caixa.setDtMov(DateHelper.getDate("2010-01-15"));  
sessionStatefull.insert(caixa);
```

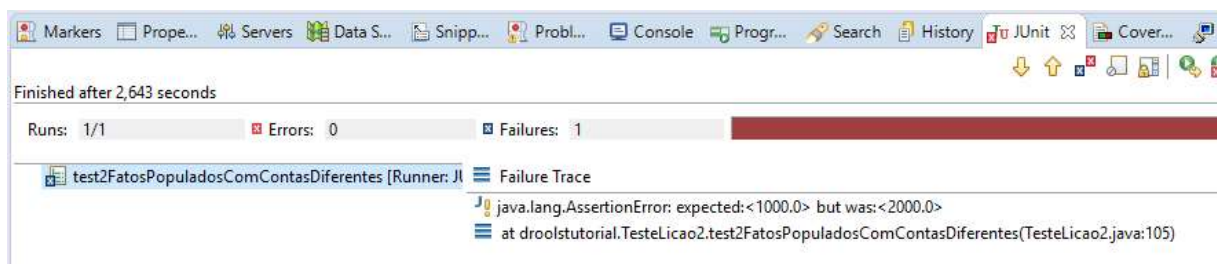
```
MovimentacaoCaixa caixa2 = new MovimentacaoCaixa();  
caixa2.setNrConta(2);  
caixa2.setTpMov(MovimentacaoCaixa.CREDITO);  
caixa2.setVIMontante(1000.0);  
caixa2.setDtMov(DateHelper.getDate("2010-01-15"));  
sessionStatefull.insert(caixa2);
```

```
sessionStatefull.fireAllRules();
```

```
Assert.assertEquals(1000.0, conta.getVIBalanco(), 0);
```

```
}
```

Execute o teste, que deve falhar.



No console você pode ver que a “Regra Crédito” foi executada duas vezes:


```

<terminated> Rerun droolstutorial.TesteLicao2.test2FatosPopuladosComContasDiferentes [JUnit] C:\Program F
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.
-----Antes-----
Objecto inserido
Conta [nrConta=1, vlBalanco=0.0]
Objecto inserido
-----Movimentação do Caixa-----)
Número da conta=1
Data do movimento= 15/01/2010
Montante movimentado=1000.0
-----Movimentação do Caixa-----)
Objecto inserido
-----Movimentação do Caixa-----)
Número da conta=2
Data do movimento= 15/01/2010
Montante movimentado=1000.0
-----Movimentação do Caixa-----)
A regra 'Regra Crédito' pode ser executada na agenda
A regra 'Regra Crédito' pode ser executada na agenda
A regra 'Regra Crédito' será executada
hora = 1523833840085 - Conta número: 1 tem agora um balanço de R$ 1000.0
A regra 'Regra Crédito' foi executada
A regra 'Regra Crédito' será executada
hora = 1523833840087 - Conta número: 1 tem agora um balanço de R$ 2000.0
A regra 'Regra Crédito' foi executada
-----depois-----
|

```

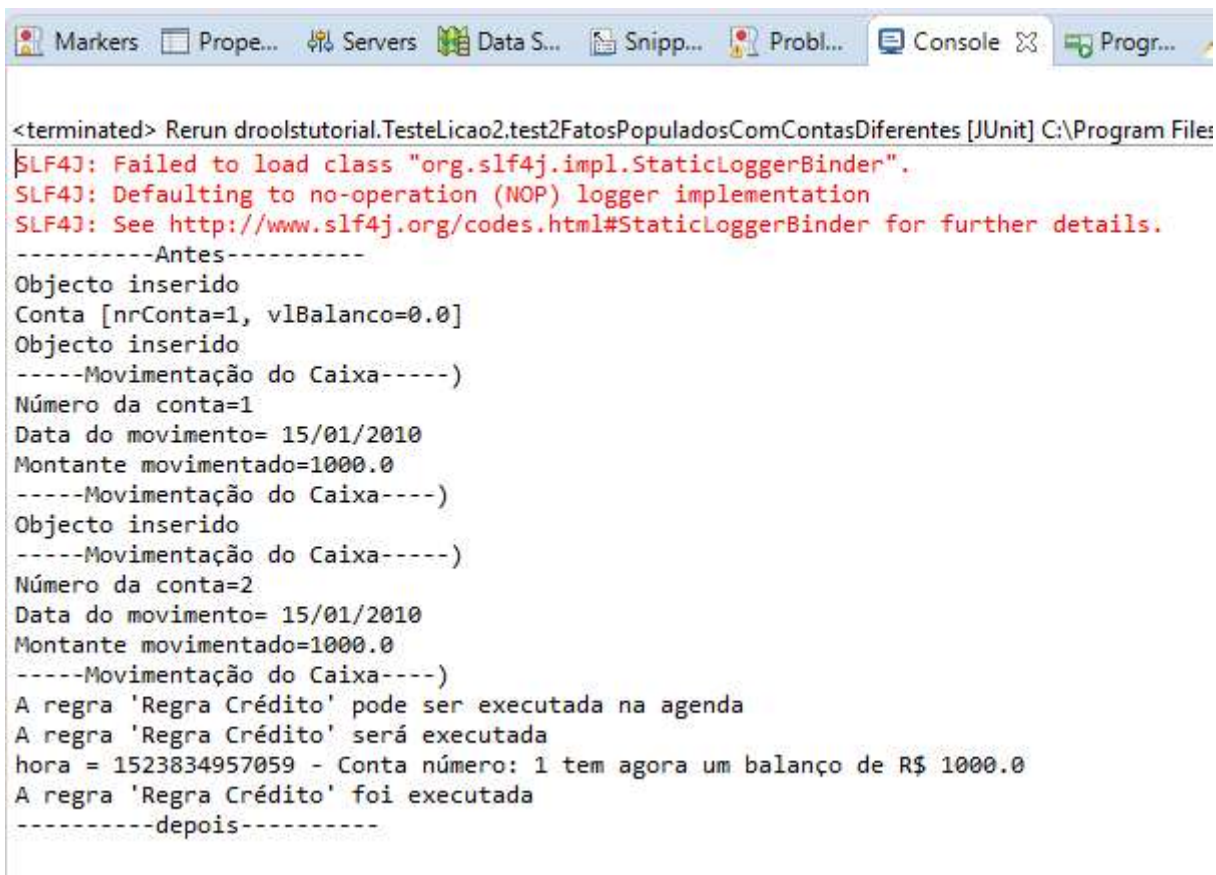
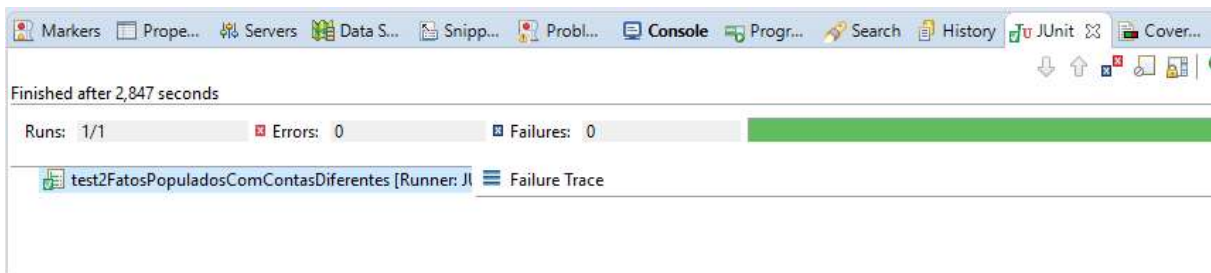
Para que a regra fique correta, é necessário vincular o atributo do número da conta dos dois fatos. Isso pode ser feito realizando as seguintes modificações:

```

rule "Regra Crédito"
  when
    $movCa: MovimentacaoCaixa( $nrConta: nrConta, tpMov == MovimentacaoCaixa.CREDITO
  )
    $conta: Conta( nrConta == $nrConta )
  then
    $conta.setVIBalanco($conta.getVIBalanco() + $movCa.getVIMontante());
    resultado.exibeTexto("Conta número: " + $conta.getNrConta() + " tem agora um balanço de
R$ " + $conta.getVIBalanco());
  end

```

A variável \$nrConta é criada dentro do movimenta da conta e o valor dela é comparada a nrConta do fato Conta. Agora, depois de melhorá-la, toda vez que você passar os fatos MovimentacaoConta (tipo Crédito) e Conta, sendo que o valor do atributo nrConta dos dois fatos sejam iguais, ligando-os, então a “Regra Crédito” será executada corretamente. A regra será executada apenas uma vez e o balanço estará correto (execute o teste novamente).



1.4.4. Calculando o balanço

Agora que você sabe como vincular os fatos usando ligação de atributos, modifique a “Regra Crédito” e crie uma “Regra Débito”, como mostrada abaixo:

```
rule "Regra Crédito"
when
    $movCa: MovimentacaoCaixa( $dtMov: dtMov, $nrConta: nrConta, tpMov == MovimentacaoCaixa.CREDITO )
    $conta: Conta( nrConta == $nrConta )
    $perCt: PeriodoContabil( dtInicio <= $dtMov && dtFim >= $dtMov )
then
    $conta.setVIBalanco($conta.getVIBalanco() + $movCa.getVIMontante());
    resultado.exibeTexto("Conta número: " + $conta.getNrConta() + " tem agora um balanço de R$ " + $conta.getVIBalanco());
```

end

rule "Regra Débito"

when

\$movCa: MovimentacaoCaixa(\$dtMov: dtMov, \$nrConta: nrConta, tpMov == MovimentacaoCaixa.DEBITO)

\$conta: Conta(nrConta == \$nrConta)

\$perCt: PeriodoContabil(dtInicio <= \$dtMov && dtFim >= \$dtMov)

then

\$conta.setVIBalanco(\$conta.getVIBalanco() - \$movCa.getVIMontante());

resultado.exibeTexto("Conta número: " + \$conta.getNrConta() + " tem agora um balanço de R\$ " + \$conta.getVIBalanco());

end

Crie um caso de teste que simule a proposta [inicial](#).

@Test

```
public void testCalculandoBalanco() throws Exception {  
    sessionStatefull = KnowledgeSessionHelper  
        .getStatefulKnowledgeSessionWithCallback(kieContainer,"ksession-licao2");
```

```
    OutputDisplay display = new OutputDisplay();  
    sessionStatefull.setGlobal("resultado", display);
```

```
    Conta conta = new Conta();  
    conta.setNrConta(1);  
    conta.setVIBalanco(0);  
    sessionStatefull.insert(conta);
```

```
    MovimentacaoCaixa caixa = new MovimentacaoCaixa();  
    caixa.setNrConta(1);  
    caixa.setTpMov(MovimentacaoCaixa.CREDITO);  
    caixa.setVIMontante(1000.0);  
    caixa.setDtMov(DateHelper.getDate("2016-01-15"));  
    sessionStatefull.insert(caixa);
```

```
    MovimentacaoCaixa caixa2 = new MovimentacaoCaixa();  
    caixa2.setNrConta(1);  
    caixa2.setTpMov(MovimentacaoCaixa.DEBITO);  
    caixa2.setVIMontante(500.0);  
    caixa2.setDtMov(DateHelper.getDate("2016-01-15"));  
    sessionStatefull.insert(caixa2);
```

```
    MovimentacaoCaixa caixa3 = new MovimentacaoCaixa();  
    caixa3.setNrConta(1);  
    caixa3.setTpMov(MovimentacaoCaixa.CREDITO);  
    caixa3.setVIMontante(1000.0);  
    caixa3.setDtMov(DateHelper.getDate("2016-04-15"));  
    sessionStatefull.insert(caixa3);
```

```
    PeriodoContabil contabil = new PeriodoContabil();  
    contabil.setDtInicio(DateHelper.getDate("2016-01-01"));  
    contabil.setDtFim(DateHelper.getDate("2016-03-31"));  
    sessionStatefull.insert(contabil);
```

```
    sessionStatefull.fireAllRules();
```

```
    Assert.assertEquals(500.0, conta.getVIBalanco(), 0);
```

```
}
```

Ao executá-lo o resultado será este:

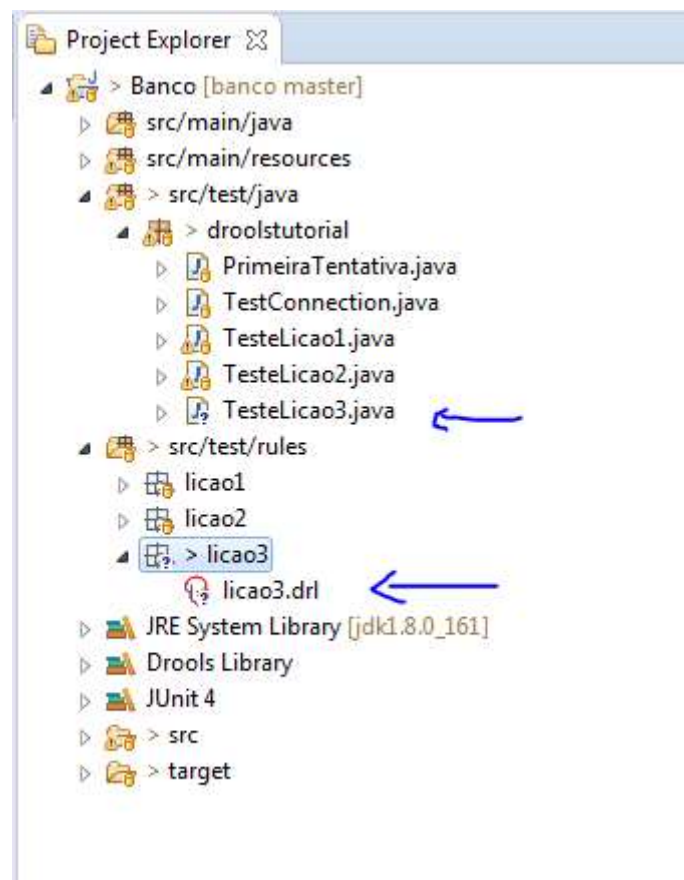
```
Markers Properties Servers Data Source Explorer Snippets Problems Console
<terminated> Rerun droolstutorial.TesteLicao2.testCalculandoBalanco [JUnit] C:\Program Files\Java\jdk1.8.0_152\bin\
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
-----Antes-----
Objecto inserido
Conta [nrConta=1, vlBalanco=0.0]
Objecto inserido
-----Movimentação do Caixa-----)
Número da conta=1
Data do movimento= 15/01/2016
Montante movimentado=1000.0
-----Movimentação do Caixa-----)
Objecto inserido
-----Movimentação do Caixa-----)
Número da conta=1
Data do movimento= 15/01/2016
Montante movimentado=500.0
-----Movimentação do Caixa-----)
Objecto inserido
-----Movimentação do Caixa-----)
Número da conta=1
Data do movimento= 15/04/2016
Montante movimentado=1000.0
-----Movimentação do Caixa-----)
Objecto inserido
PeriodoContabil [dtInicio=Fri Jan 01 00:00:00 BRST 2016, dtFim=Thu Mar 31 00:00:00 BRT 2016]
A regra 'Regra Crédito' pode ser executada na agenda
A regra 'Regra Crédito' será executada
hora = 1523836092143 - Conta número: 1 tem agora um balanço de R$ 1000.0
A regra 'Regra Crédito' foi executada
A regra 'Regra Débito' pode ser executada na agenda
A regra 'Regra Débito' será executada
hora = 1523836092149 - Conta número: 1 tem agora um balanço de R$ 500.0
A regra 'Regra Débito' foi executada
-----depois-----
```

Como esperado, a “Regra Crédito” foi executada uma vez e a “Regra Débito” também foi executada uma vez. O movimento do dia 15 de abril foi ignorado porque não satisfaz a validação do período contábil estipulado pela regra.

1.5. Drools language: Exemplos de uso.

Na primeira lição você entendeu como a engenharia da regra funciona. Na segunda lição você começou a estudar como funciona a ligação dos fatos em uma regra. Nesta lição, você estudará um pouco mais das possibilidades que a ferramenta oferece para escrever regras em casos mais complexos.

1. Crie uma classe TesteLicao3.java, um pacote licao3 e um arquivo licao3.drl apontado para o pacote droolstutorial, como feito nas lições anteriores. O projeto deverá ficar assim:



```
package droolstutorial;
```

```
import org.junit.After;
import org.junit.Before;
import org.junit.BeforeClass;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.StatelessKieSession;
```

```
import util.KnowledgeSessionHelper;
```

```
public class TesteLicao3 {
    static KieContainer kieContainer;
```

```

StatelessKieSession sessionStateless = null;
KieSession sessionStatefull = null;

@BeforeClass
public static void beforeClass() {
    kieContainer = KnowledgeSessionHelper.criarRegraBase();
}

@Before
public void antes() {
    System.out.println("-----Antes-----");
}

@After
public void depois() {
    System.out.println("-----depois-----");
}
}

```

2. Acrescente uma nova sessão no kmodule.xml.

```

<kbase name="rules3" packages="licao3">
    <ksession name="ksession-licao3"/>
</kbase>

```

Ao criar os exemplos, você verá mais regras disparadas do que exemplos mostrados. Como o Drools é uma linguagem declarativa, assim que a restrição é satisfeita, a regra pode disparar.

3. Para ver os exemplos mais avançados, adicione no pacote droolstutorial as classes Cliente.java e ContaPrivada.java.

```

package droolstutorial;

public class Cliente {

    private String nome;
    private String sobrenome;
    private String pais;

    public Cliente(String nome, String sobrenome, String pais) {
        super();
        this.nome = nome;
        this.sobrenome = sobrenome;
        this.pais = pais;
    }

    public Cliente() {
        super();
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}

```

```

    }

    public String getSobrenome() {
        return sobrenome;
    }

    public void setSobrenome(String sobrenome) {
        this.sobrenome = sobrenome;
    }

    public String getPais() {
        return pais;
    }

    public void setPais(String pais) {
        this.pais = pais;
    }

    @Override
    public String toString() {
        StringBuffer buff = new StringBuffer();
        buff.append("-----Cliente-----\n");
        buff.append("Nome=" + this.nome + "\n");
        buff.append("Sobrenome=" + this.sobrenome + "\n");
        buff.append("País=" + this.pais + "\n");
        buff.append("-----Cliente fim-");
        return buff.toString();
    }
}

```

```
package droolstutorial;
```

```

public class ContaPrivada extends Conta{

    private Cliente dono;

    public Cliente getDono() {
        return dono;
    }

    public void setDono(Cliente dono) {
        this.dono = dono;
    }

    @Override
    public String toString() {
        StringBuffer buff = new StringBuffer();
        buff.append("-----Conta Privada-----\n");
        buff.append(super.toString());
        if (this.dono != null) {
            buff.append("Dono da conta: " + this.dono.toString() + "\n");
        }
        buff.append("-----Conta Privada fim-");
        return buff.toString();
    }
}

```


1.5.1. Contraint

Validando um atributo em uma lista de valores. Verificando se o tipo de uma conta é crédito ou débito.

```
//created on: 19/04/2018
package droolstutorial

//list any import classes here.
import droolstutorial.Cliente;
import droolstutorial.ContaPrivada;
import droolstutorial.MovimentacaoCaixa;
import util.OutputDisplay;

//declare any global variables here
global OutputDisplay resultado;

rule "A movimentação do caixa pode ser Crédito ou Débito"
when
    $caixa : MovimentacaoCaixa(tpMov in ( MovimentacaoCaixa.DEBITO , MovimentacaoCaixa.CREDITO)
)

then
    resultado.exibeTexto("A movimentação do caixa pode ser Crédito ou Débito");
end

@Test
public void testInConstrait() throws Exception {
    sessionStatefull = KnowledgeSessionHelper
        .getStatefulKnowledgeSessionWithCallback(kieContainer,"ksession-licao3");

    OutputDisplay display = new OutputDisplay();
    sessionStatefull.setGlobal("resultado", display);

    MovimentacaoCaixa caixa = new MovimentacaoCaixa();
    caixa.setTpMov(MovimentacaoCaixa.CREDITO);
    sessionStatefull.insert(caixa);

    sessionStatefull.fireAllRules();
}
```

No console deverá aparecer assim:

```

Markers Properties Servers Data Source Explorer Snippets Console JUnit Progress
<terminated> TesteLicao3.testInConstrait [JUnit] C:\Program Files\Java\jdk1.8.0_161\bin\javaw.exe (19 de abr de 2018 13:32:39)
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.
-----Antes-----
Objecto inserido
-----Movimentação do Caixa-----)
Número da conta=0
Nenhuma data de movimento setada
Montante movimentado=0.0
-----Movimentação do Caixa----)
A regra 'A movimentação do caixa pode ser Crédito ou Débito' pode ser executada na agenda
A regra 'A movimentação do caixa pode ser Crédito ou Débito' será executada
hora = 1524155561450 - A movimentação do caixa pode ser Crédito ou Débito
A regra 'A movimentação do caixa pode ser Crédito ou Débito' foi executada
-----depois-----

```

1.5.2. Acessor

Isso permite adicionar uma restrição a uma classe de atributo sem a necessidade de adicionar o objeto vinculado à sessão.

rule "Accessor"

when

\$caixa: ContaPrivada(dono.nome == "João")

then

resultado.exibeTexto("Conta pertence a João");

end

@Test

public void testAccessor() **throws** Exception {

sessionStatefull = KnowledgeSessionHelper

.getStatefulKnowledgeSessionWithCallback(kieContainer, "ksession-licao3");

OutputDisplay display = **new** OutputDisplay();

sessionStatefull.setGlobal("resultado", display);

Cliente cliente = **new** Cliente();

cliente.setNome("João");

cliente.setSobrenome("Ferreira");

ContaPrivada contaPrivada = **new** ContaPrivada();

contaPrivada.setDono(cliente);

sessionStatefull.insert(contaPrivada);

sessionStatefull.fireAllRules();

}

Ao executar, você verá que o cliente não foi adicionado a instancia da sessão do Drools.


```

<terminated> TesteLicao3.testAccessor [JUnit] C:\Program Files\Java\jdk1.8.0_161\bin\javaw.exe (19 de abr de 2018)
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.
-----Antes-----
Objecto inserido
-----Conta Privada-----)
Conta [nrConta=0, vlBalanco=0.0]Dono da conta: -----Cliente-----)
Nome=João
Sobrenome=Ferreira
País=null
-----Cliente fim-)
-----Conta Privada fim-)
A regra 'Accessor' pode ser executada na agenda
A regra 'Accessor' será executada
hora = 1524156344499 - Conta pertence a João
A regra 'Accessor' foi executada
-----depois-----

```

1.5.3. And/or

É possível restringir um elemento similar ao que é feito em java.

```

rule "infixAnd"
when
    ($c1 : Cliente ( pais == "GB") and ContaPrivada( dono == $c1 ) )
    or
    ($c1 : Cliente ( pais == "US") and ContaPrivada( dono == $c1 ) )
then
    resultado.exibeTexto("Pessoa vive em GB ou US");
end

```

```

@Test
public void testInOrFact() throws Exception {
    sessionStatefull = KnowledgeSessionHelper
        .getStatefulKnowledgeSessionWithCallback(kieContainer, "ksession-licao3");
    OutputDisplay display = new OutputDisplay();
    sessionStatefull.setGlobal("resultado", display);
    Cliente cliente = new Cliente();

    cliente.setPais("GB");
    sessionStatefull.insert(cliente);

    ContaPrivada contaPrivada = new ContaPrivada();
    contaPrivada.setDono(cliente);
    sessionStatefull.insert(contaPrivada);

    sessionStatefull.fireAllRules();
}

```

```

<terminated> TesteLicao3.testInOrFact [JUnit] C:\Program Files\Java\jdk1.8.0_161\bin\javaw.exe (19 de abr de 201
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.
-----Antes-----
Objecto inserido
-----Cliente-----)
Nome=null
Sobrenome=null
País=GB
-----Cliente fim-)
Objecto inserido
-----Conta Privada-----)
Conta [nrConta=0, vlBalanço=0.0]Dono da conta: -----Cliente-----)
Nome=null
Sobrenome=null
País=GB
-----Cliente fim-)
-----Conta Privada fim-)
A regra 'infixAnd' pode ser executada na agenda
A regra 'infixAnd' será executada
hora = 1524157034385 - Pessoa vive em GB ou US
A regra 'infixAnd' foi executada
-----depois-----

```

1.5.4. Not

É possível testar se um tipo de fato não foi passado na sessão.

```

rule "Não eh Cliente"
when
    not Cliente( )
then
    resultado.exibeTexto("Não é Cliente");
end

```

```

@Test
public void testNotCondition() throws Exception {
    sessionStatefull = KnowledgeSessionHelper
        .getStatefulKnowledgeSessionWithCallback(kieContainer, "ksession-licao3");

    OutputDisplay display = new OutputDisplay();
    sessionStatefull.setGlobal("resultado", display);

    sessionStatefull.fireAllRules();
}

```

```
Markers Properties Servers Data Source Explorer Snippets JUnit Progress
<terminated> TesteLicao3.testNotCondition [JUnit] C:\Program Files\Java\jdk1.8.0_161\bin\javaw.exe (19 de abr
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.
-----Antes-----
A regra 'Não eh Cliente' pode ser executada na agenda
A regra 'Não eh Cliente' será executada
hora = 1524157559791 - Não é Cliente
A regra 'Não eh Cliente' foi executada
-----depois-----
```

1.5.5. Exists

Ao contrário do teste anterior, é possível testar se existe um fato na sessão.

```
rule "Existe"
when
    exists Conta( )
then
    resultado.exibeTexto("Existe uma Conta");
end
```

```
@Test
public void testExistsCondition() throws Exception {
    sessionStatefull = KnowledgeSessionHelper
        .getStatefulKnowledgeSessionWithCallback(kieContainer, "ksession-licao3");

    OutputDisplay display = new OutputDisplay();
    sessionStatefull.setGlobal("resultado", display);

    Conta conta = new Conta();
    sessionStatefull.insert(conta);

    Cliente cliente = new Cliente();
    sessionStatefull.insert(cliente);

    sessionStatefull.fireAllRules();
}
```

```

<terminated> TesteLicao3.testExistsCondition [JUnit] C:\Program Files\Java\jdk1.8.0_161\bin\javaw.exe (19 de abr 2017)
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.
-----Antes-----
Objecto inserido
Conta [nrConta=0, vlBalanco=0.0]
Objecto inserido
-----Cliente-----)
Nome=null
Sobrenome=null
País=null
-----Cliente fim-----
A regra 'Existe' pode ser executada na agenda
A regra 'Existe' será executada
hora = 1524157820023 - Existe uma Conta
A regra 'Existe' foi executada
-----depois-----

```

1.5.6. ForAll

Verifica se cada instância de MovimentoConta está vinculado a uma instância de Conta.

```

rule "ForAll"
when
    forall ( Conta( $nrConta : nrConta ) MovimentacaoCaixa( nrConta == $nrConta ) )
then
    resultado.exibeTexto("Toda a movimentação do caixa está vinculada a uma conta");
end

```

Nesta regra, na condição “forall”, a instância de MovimentacaoCaixa está a instância Conta. Veja um teste onde todos os objetos estão relacionados.

```

@Test
public void testForAll() throws Exception {
    sessionStatefull = KnowledgeSessionHelper
        .getStatefulKnowledgeSessionWithCallback(kieContainer, "ksession-licao3");

    OutputDisplay display = new OutputDisplay();
    sessionStatefull.setGlobal("resultado", display);

    Conta conta = new Conta();
    conta.setNrConta(1);
    conta.setVlBalanco(0);
    sessionStatefull.insert(conta);

    MovimentacaoCaixa caixa = new MovimentacaoCaixa();
    caixa.setNrConta(1);
    sessionStatefull.insert(caixa);

    MovimentacaoCaixa caixa2 = new MovimentacaoCaixa();
    caixa2.setNrConta(1);
    sessionStatefull.insert(caixa2);
}

```

```

    sessionStatefull.fireAllRules();
}

```

Ao executar o teste, a regra será executada.

```

A regra 'ForAll' pode ser executada na agenda
A regra 'ForAll' será executada
hora = 1524158805773 - Toda a movimentação do caixa está vinculada a uma conta
A regra 'ForAll' foi executada

```

A regra não executará se o trecho abaixo for acrescentado ao teste:

```

Conta conta2 = new Conta();
conta.setNrConta(2);
conta.setVIBalanco(0);
sessionStatefull.insert(conta2);

MovimentacaoCaixa caixa3 = new MovimentacaoCaixa();
caixa3.setNrConta(2);
sessionStatefull.insert(caixa3);

sessionStatefull.fireAllRules();

```

1.5.7. From

Algumas vezes é necessário acessar dados que estão fora da sessão do Drools. Como não é possível inserir todos os objetos na sessão, para chama-los pode usar a instrução “from” na parte “when”.

Crie a classe ClienteServices dentro do pacote droolstutorial.service.

```

package droolstutorial.service;

import java.util.ArrayList;
import java.util.List;

import droolstutorial.Cliente;

public class ClienteService {

    public List<Cliente> getListCliente() {
        List<Cliente> result = new ArrayList<Cliente>();
        result.add(new Cliente("João", "Ferreira", "Fr"));
        result.add(new Cliente("João", "Ferraz", "GB"));
        result.add(new Cliente("João", "Ferreira", "GB"));

        return result;
    }
}

```

Crie a regra que usa o from, importando o fato ClienteService e a variável global ClienteService.

```

//list any import classes here.
import droolstutorial.service.ClienteService;

//declare any global variables here

```



```
global ClienteService clienteService;
```

```
[...]
```

```
rule "FromCondition"
```

```
when
```

```
    $c : Cliente()
```

```
    $cc : Cliente(nome == $c.nome , sobrenome == $c.sobrenome, pais != $c.pais) from clienteService.getListCliente();
```

```
then
```

```
    resultado.exibeTexto("Encontrado o mesmo cliente em dois países");
```

```
end
```

```
@Test
```

```
public void testFromLHS() throws Exception {  
    sessionStatefull = KnowledgeSessionHelper  
        .getStatefulKnowledgeSessionWithCallback(kieContainer, "ksession-licao3");
```

```
    OutputDisplay display = new OutputDisplay();  
    sessionStatefull.setGlobal("resultado", display);
```

```
    sessionStatefull.setGlobal("clienteService", new ClienteService());  
    Cliente cliente = new Cliente("João", "Ferreira", "Br");  
    sessionStatefull.insert(cliente);
```

```
    sessionStatefull.fireAllRules();
```

```
}
```

A regra foi executada duas vezes mostrando que o serviço possui dois clientes com o mesmo nome, porém em países diferentes.

```
A regra 'FromCondition' pode ser executada na agenda  
A regra 'FromCondition' pode ser executada na agenda  
A regra 'FromCondition' será executada  
hora = 1524160491414 - Encontrado os mesmos clientes em dois países  
A regra 'FromCondition' foi executada  
A regra 'FromCondition' será executada  
hora = 1524160491414 - Encontrado os mesmos clientes em dois países  
A regra 'FromCondition' foi executada
```

1.5.8. Collecting

O objetivo é coletar um conjunto de fatos e restrições, se as restrições forem verdadeiras. Na regra de exemplo, serão coletados todos os MovimentacaoCaixa que estão no período de tempo correto e pertence a uma conta passada. A sintaxe "from collect" retorna um ArrayList(). É possível adicionar uma condição como na primeira regra em que foi adicionada uma restrição que esperava pelo menos dois itens. Na segunda regra, não será adicionada essa restrição.

Importe o ArrayList().

//list any import classes here.

import java.util.ArrayList

[...]

rule "Mais de 2 Linhas de Movimentação do Caixa"

when

\$c: Conta(\$nrConta : nrConta)

\$p: PeriodoContabil(\$dtInicio : dtInicio , \$dtFim : dtFim)

\$number: ArrayList(size >= 2)

from collect(MovimentacaoCaixa(dtMov >= \$dtInicio && dtMov <= \$dtFim , nrConta == \$nrConta

))

then

resultado.exibeTexto("Encontrado mais de duas linhas de movimentação do caixa");

resultado.exibeTexto("<<<<<<<<<<");

for (Object ff : \$number){

resultado.exibeTexto(ff.toString());

}

resultado.exibeTexto(">>>>>>>>>>>>>>>");

end

rule "Número de linhas de Movimentação do Caixa"

when

\$c: Conta(\$nrConta : nrConta)

\$p: PeriodoContabil (\$dtInicio : dtInicio , \$dtFim : dtFim)

\$number: ArrayList()

from collect(MovimentacaoCaixa(dtMov >= \$dtInicio && dtMov <= \$dtFim , nrConta == \$nrConta

))

then

resultado.exibeTexto("Encontrado "+\$number.size()+" linhas != 'Mais de 2 Linhas de Movimentação do Caixa'");

end

Acrescente os construtores abaixo as classes MovimentacaoCaixa e PeriodoContabil.

```
public MovimentacaoCaixa() {
```

```
    super();
```

```
    // TODO Auto-generated constructor stub
```

```
}
```

```
public MovimentacaoCaixa(Date dtMov, double vlMontante, int tpMov, long nrConta) {
```

```
    super();
```

```
    this.dtMov = dtMov;
```

```
    this.vlMontante = vlMontante;
```

```
    this.tpMov = tpMov;
```

```
    this.nrConta = nrConta;
```

```
}
```

```
public PeriodoContabil(Date dtInicio, Date dtFim) {
```

```
    super();
```

```
    this.dtInicio = dtInicio;
```

```
    this.dtFim = dtFim;
```

```
}
```

```
public PeriodoContabil() {}
```

E o caso de teste:

```
@Test
public void testCollecting() throws Exception {
    sessionStatefull = KnowledgeSessionHelper
        .getStatefullKnowledgeSessionWithCallback(kieContainer, "ksession-licao3");
}
```

```
OutputDisplay display = new OutputDisplay();
sessionStatefull.setGlobal("resultado", display);
```

```
Conta conta = new Conta();
conta.setNrConta(1);
conta.setVIBalanco(0);
sessionStatefull.insert(conta);
```

```

        sessionStatefull.insert(new MovimentacaoCaixa(DateHelper.getDate("2010-01-15"), 1000, MovimentacaoCaixa.CREDITO, 1));
        sessionStatefull.insert(new MovimentacaoCaixa(DateHelper.getDate("2010-02-15"), 500, MovimentacaoCaixa.DEBITO, 1));
        sessionStatefull.insert(new MovimentacaoCaixa(DateHelper.getDate("2010-04-15"), 1000, MovimentacaoCaixa.CREDITO, 1));
        sessionStatefull.insert(new PeriodoContabil(DateHelper.getDate("2010-01-01"), DateHelper.getDate("2010-31-31")));

        sessionStatefull.fireAllRules();
    }
}

```

```
A regra 'Mais de 2 Linhas de Movimentação do Caixa' pode ser executada na agenda
A regra 'Mais de 2 Linhas de Movimentação do Caixa' será executada
hora = 1524165010162 - Encontrado mais de duas linhas de movimentação do caixa
hora = 1524165010162 - <<<<<<<<
hora = 1524165010162 - -----Movimentação do Caixa-----)
Número da conta=1
Data do movimento= 15/04/2010
Montante movimentado=1000.0
----Movimentação do Caixa----

```
hora = 1524165010162 - -----Movimentação do Caixa-----)
Número da conta=1
Data do movimento= 15/02/2010
Montante movimentado=500.0
----Movimentação do Caixa----
```



```
hora = 1524165010162 - -----Movimentação do Caixa-----)
Número da conta=1
Data do movimento= 15/01/2010
Montante movimentado=1000.0
----Movimentação do Caixa----
```



```
hora = 1524165010162 - >>>>>>>>>>>>
```



A regra 'Mais de 2 Linhas de Movimentação do Caixa' foi executada  
A regra 'Número de linhas de Movimentação do Caixa' pode ser executada na agenda  
A regra 'Número de linhas de Movimentação do Caixa' será executada  
hora = 1524165010178 - Encontrado 3 linhas != 'Mais de 2 Linhas de Movimentação do Caixa'  
A regra 'Número de linhas de Movimentação do Caixa' foi executada


```

1.5.9. Accumulating

Na seção anterior, dados foram coletados. Existe o “**from accumulate**” que permite somar dados em um comando.

O “**from accumulate**” recebe 5 parâmetros:

- a) Uma expressão de restrição de um fato;
- b) Uma condição inicial;
- c) A instrução “when” da regra aplica a expressão da restrição do fato;
- d) Na ação inversa “when” a expressão de restrição de fato não é mais verdade;
- e) O resultado do Accumulate.

rule "Regra Crédito e Débito"

when

\$c : Conta(\$nrConta : nrConta)

\$p : PeriodoContabil(\$dtInicio : dtInicio , \$dtFim : dtFim)

\$totalCredito : Number(doubleValue > 100)

from accumulate(MovimentacaoCaixa(tpMov == MovimentacaoCaixa.CREDITO,
\$vlMontante : vlMontante, dtMov >= \$dtInicio && dtMov <= \$dtFim, nrConta == \$nrConta),

init(**double** total = 0;),

action(total += \$vlMontante;),

reverse(total -= \$vlMontante;),

result(total))

\$totalDebito : Number(doubleValue > 100)

from accumulate(MovimentacaoCaixa(tpMov == MovimentacaoCaixa.DEBITO,
\$vlMontante : vlMontante, dtMov >= \$dtInicio && dtMov <= \$dtFim , nrConta == \$nrConta),

init(**double** total = 0;),

action(total += \$vlMontante;),

reverse(total -= \$vlMontante;),

result(total))

then

resultado.exibeTexto(" Encontrado "+\$totalCredito+" como um crédito");

resultado.exibeTexto(" Encontrado "+\$totalDebito+" como um débito");

end

A restrição aqui está em um tipo de fato MovimentacaoCaixa com as restrições que já utilizamos antes (é vinculado a uma conta e o período contábil) que deve ser crédito ou débito.

Então a condição inicial é inicializada (“init”) com um valor duplo que foi chamada de “total”. Então, na ação/reversão, adiciona ao total o valor do montante no Movimentacao-Caixa obtida usando uma ligação de atributo. Na ação resultante, retorna o valor calculado.

@Test

```
public void testAccumulate() throws Exception {  
    sessionStatefull = KnowledgeSessionHelper  
        .getStatefulKnowledgeSessionWithCallback(kieContainer, "ksession-licao3");
```

```
    OutputDisplay display = new OutputDisplay();  
    sessionStatefull.setGlobal("resultado", display);
```

```
    Conta conta = new Conta();  
    conta.setNrConta(1);  
    conta.setVIBalanco(0);  
    sessionStatefull.insert(conta);
```

```
    FactHandle fa = sessionStatefull.insert(new MovimentacaoCaixa(DateHelper.getDate("2010-01-15"),  
1000, MovimentacaoCaixa.CREDITO, 1));
```

```
        sessionStatefull.insert(new MovimentacaoCaixa(DateHelper.getDate("2010-02-15"), 500, Movimenta-
caoCaixa.DEBITO, 1));
        sessionStatefull.insert(new MovimentacaoCaixa(DateHelper.getDate("2010-04-15"), 1000, Movimenta-
caoCaixa.CREDITO, 1));
        sessionStatefull.insert(new PeriodoContabil(DateHelper.getDate("2010-01-01"), DateHelper.ge-
tDate("2010-12-31")));

        sessionStatefull.fireAllRules();

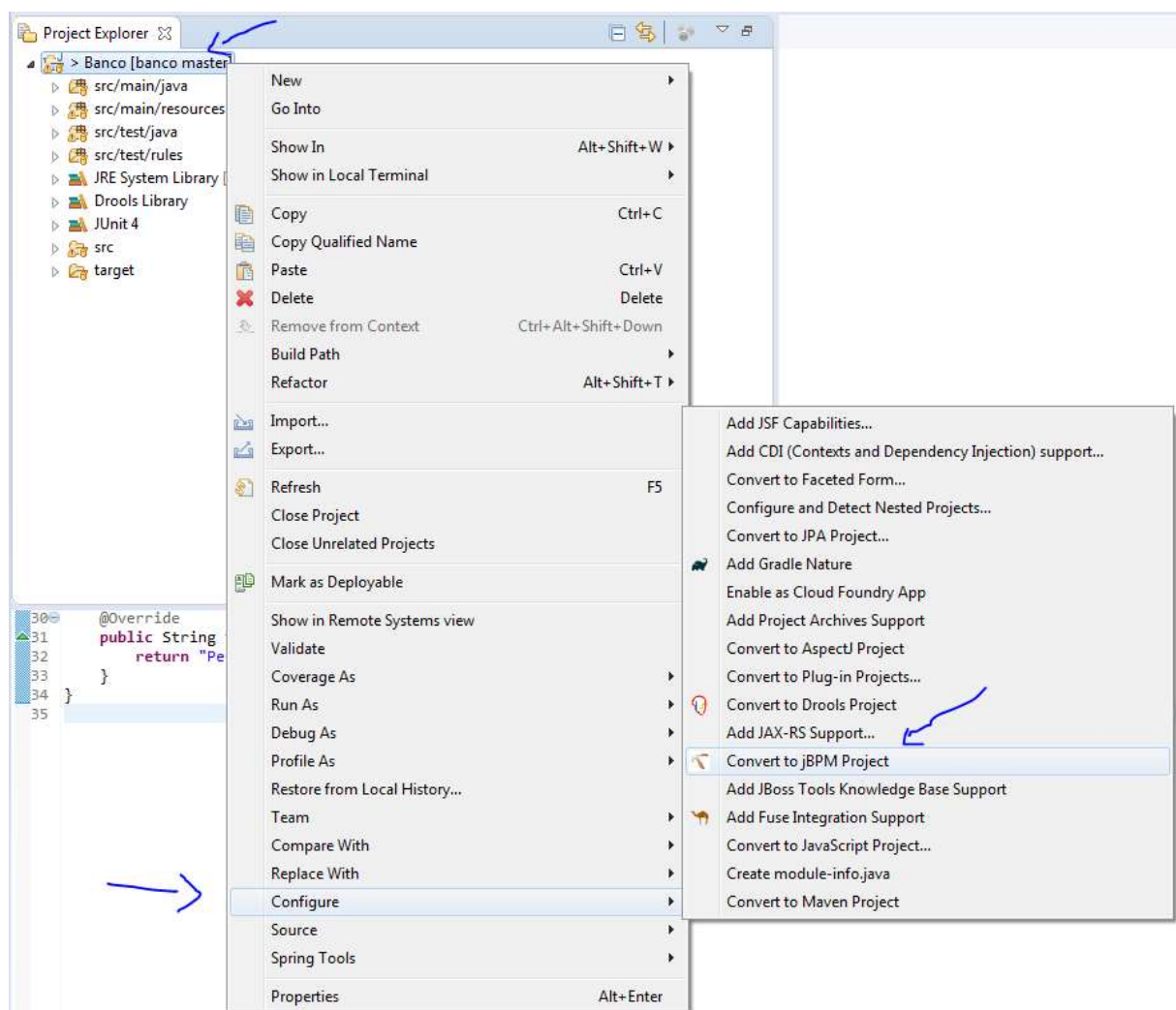
        sessionStatefull.delete(fa);

        sessionStatefull.fireAllRules();
    }
```

1.6. Fluxo de regras: organizando a execução de regras em projetos maiores

Ao capturar os requisitos de negócios, a maioria dos usuários expressa as regras dividindo o problema para resolver em etapas. Por isso, é muito conveniente poder implementá-lo da mesma maneira. Sabendo disso, na tecnologia Drools/jBPM, um processo jBPM usa etapas de regra, e isso é chamado de fluxo de regra. Onde, na realidade, é possível unir o Drools com o jBPM.

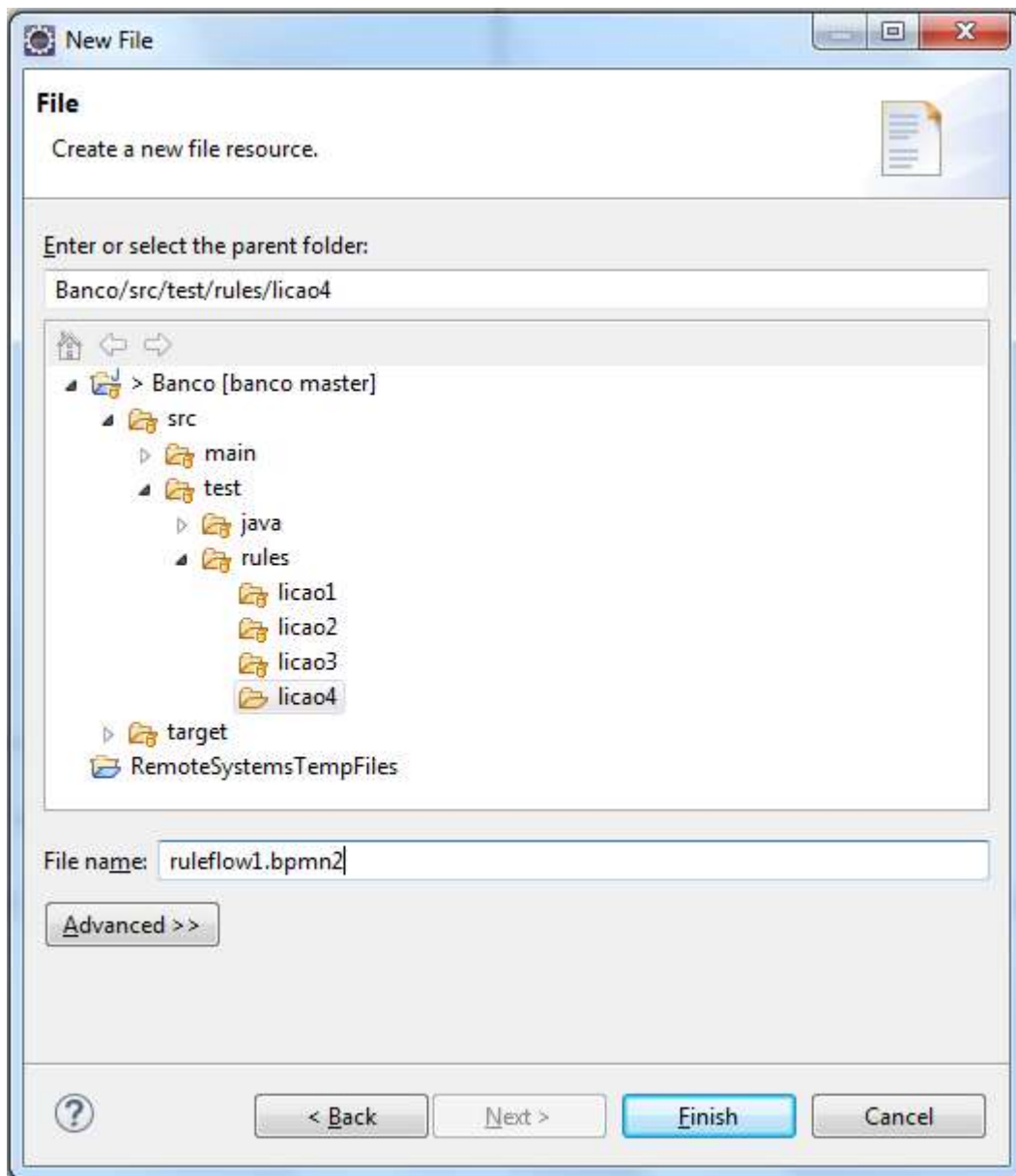
Antes, configure o projeto para se tornar um projeto jBPM: Selecione o projeto > Click botão direito > configure > convert to jBPM Project.

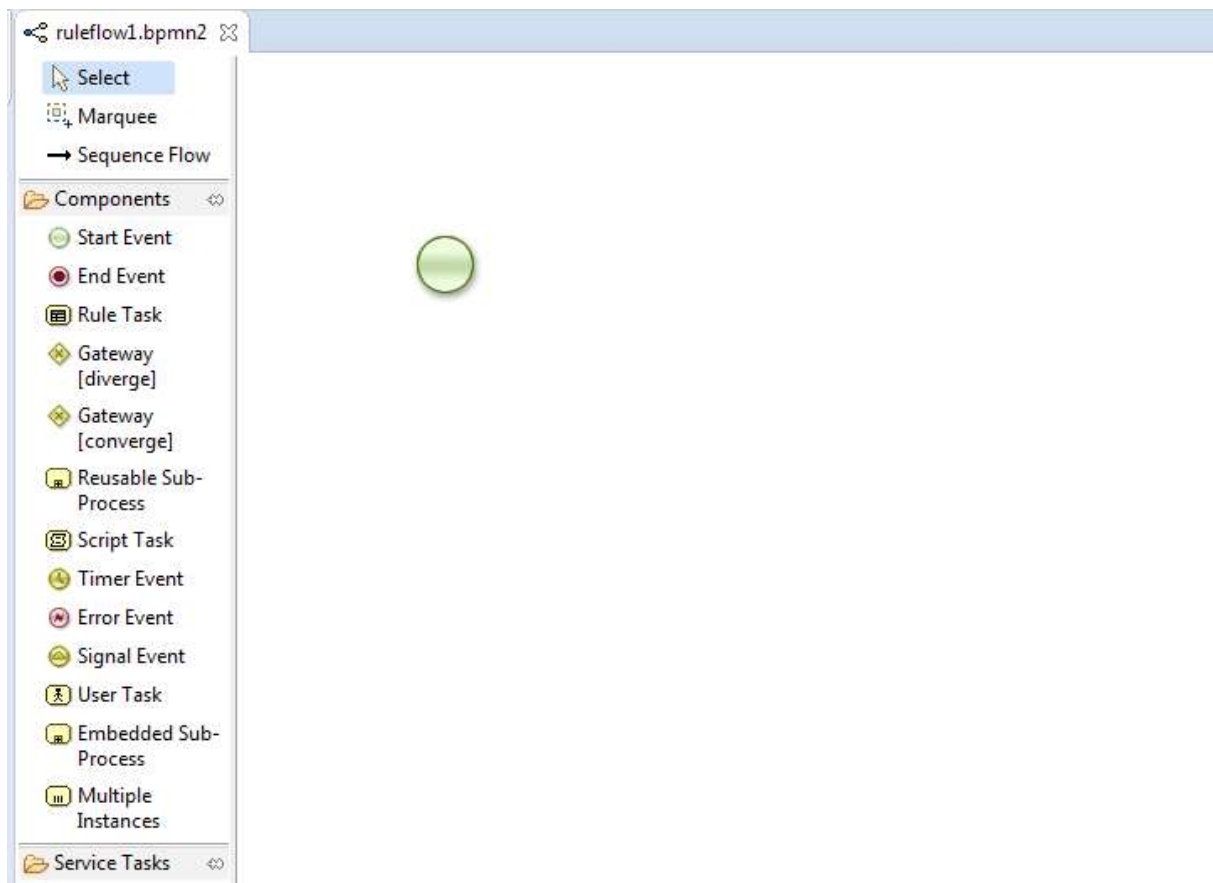


1.6.1. Criando seu primeiro fluxo de regras

Crie o pacote licao4, dentro dele crie um novo arquivo e chame de ruleflow1.bpmn2 e clique Ok. Uma mensagem de erro irá aparecer que o arquivo está vazio, mas o plugin de evento inicial será criado.

New > Other > File > Next > ruleflow2.bpmn2 > Finish.





Acrescente duas tarefas de regras e um evento final.



Então, selecione cada uma das tarefas e coloque as propriedades como abaixo:

Property	Value
Id	2
MetaData	{Uniqueld=0e9beb1b-70be-46ef-aff0-328364f4b064, x=212, width=80, y=99, height=48}
Name	verificar
RuleFlowGroup	grupo1
Timers	

Property	Value
Id	3
MetaData	{Uniqueld=829ec7f5-d737-45ca-a23d-1233fee58b1a, x=376, width=80, y=100, height=48}
Name	calcular
RuleFlowGroup	grupo2
Timers	

As propriedades do workflow (o background branco) ficarão assim:

Property	Value
Connection Layout	Manual
Exception Handlers	{}
Id	RF1
Name	ruleflow1
Package	curso
Swimlanes	[]
Variables	[]
Version	

Crie um novo arquivo de regras (.drl) e chame de ruleflow1.drl também dentro do pacote licao4, no Rule package name coloque "droolstutorial".

```
//created on: 19/04/2018
```

```
package droolstutorial
```

```
//list any import classes here.
```

```
import droolstutorial.Conta;
```

```
import droolstutorial.MovimentacaoCaixa;
```

```
import droolstutorial.PeriodoContabil;
```

```
import util.OutputDisplay;
```

```
//declare any global variables here
```

```
global OutputDisplay resultado;
```

```
rule "Conta grupo1"
```

```
  ruleflow-group "grupo1"
```

```
  when
```

```
    Conta( )
```

```
  then
```

```
    resultado.exibeTexto("Conta no grupo1");
```

```
end
```

```
rule "Conta grupo2"
```

```
  ruleflow-group "grupo2"
```

```
  when
```

```
    Conta( )
```

```
  then
```

```
    resultado.exibeTexto("Conta no grupo2");
```

```
end
```

Não esqueça de adicionar uma nova entrada no kmodule.xml.

```
<kbase name="rules4" packages="licao4">
  <ksession name="ksession-licao4"/>
</kbase>
```

Veja a palavra-chave “ruleflow-group”. Aqui a primeira regra recebe o nome “grupo1” e a segunda regra o nome “grupo2”, estes são os mesmos nomes atribuídos aos itens definidos no processo do fluxo definido acima. Sendo assim, a primeira regra só pode ser executada se o ruleflow-group “grupo1” estiver ativado e o mesmo acontece para o segundo.

Antes de rodar o próximo teste, adicione o novo call-back na classe KnowledgeSessionHelper, para monitorar as atividades que envolvem o processo do jBPM.

```
public static KieSession getStatefulKnowledgeSessionForJBPM(
    KieContainer kieContainer, String sessionName) {
    KieSession session = getStatefulKnowledgeSessionWithCallback(kieContainer, sessionName);
    session.addEventListener(new ProcessEventListener() {

        @Override
        public void beforeVariableChanged(ProcessVariableChangedEvent arg0) {}

        @Override
        public void beforeProcessStarted(ProcessStartedEvent arg0) {
            System.out.println("Processo: \"\"+arg0.getProcessInstance().getProcessName()+\"\" foi iniciado");
        }

        @Override
        public void beforeProcessCompleted(ProcessCompletedEvent arg0) {}

        @Override
        public void beforeNodeTriggered(ProcessNodeTriggeredEvent arg0) {}

        @Override
        public void beforeNodeLeft(ProcessNodeLeftEvent arg0) {
            if (arg0.getNodeInstance() instanceof RuleSetNodeInstance){
                System.out.println("Noh: \"\"+ arg0.getNodeInstance().getNodeName()+\"\" foi deixado");
            }
        }

        @Override
        public void afterVariableChanged(ProcessVariableChangedEvent arg0) {}

        @Override
        public void afterProcessStarted(ProcessStartedEvent arg0) {}

        @Override
        public void afterProcessCompleted(ProcessCompletedEvent arg0) {
            System.out.println("Processo: \"\"+arg0.getProcessInstance().getProcessName()+\"\" parou");
        }

        @Override
```

```

    public void afterNodeTriggered(ProcessNodeTriggeredEvent arg0) {
        if (arg0.getNodeInstance() instanceof RuleSetNodeInstance){
            System.out.println("Noh \''+ arg0.getNodeInstance().getNodeName()+"\' foi inserido");
        }
    }

    @Override
    public void afterNodeLeft(ProcessNodeLeftEvent arg0) {}
});
return session;
}

```

Observe que está procurando apenas o Nó do tipo Rule Step chamado RuleSetNodeInstance. E o caso de teste será assim: crie a classe TesteLicao4.java.

```

package droolstutorial;

import util.OutputDisplay;
import org.junit.After;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import util.KnowledgeSessionHelper;

public class TesteLicao4 {
    static KieContainer kieContainer;
    KieSession sessionStatefull = null;

    @BeforeClass
    public static void beforeClass() {
        kieContainer = KnowledgeSessionHelper.criarRegraBase();
    }
    @Before
    public void setUp() throws Exception {
        System.out.println("-----Before-----");
    }
    @After
    public void tearDown() throws Exception {
        System.out.println("-----After-----");
    }

    @Test
    public void testRuleFlow1() {
        sessionStatefull = KnowledgeSessionHelper
            .getStatefulKnowledgeSessionForJBPM(kieContainer, "ksession-licao4");

        OutputDisplay display = new OutputDisplay();
        sessionStatefull.setGlobal("resultado", display);

        Conta conta = new Conta();
        sessionStatefull.insert(conta);

        sessionStatefull.startProcess("RF1");

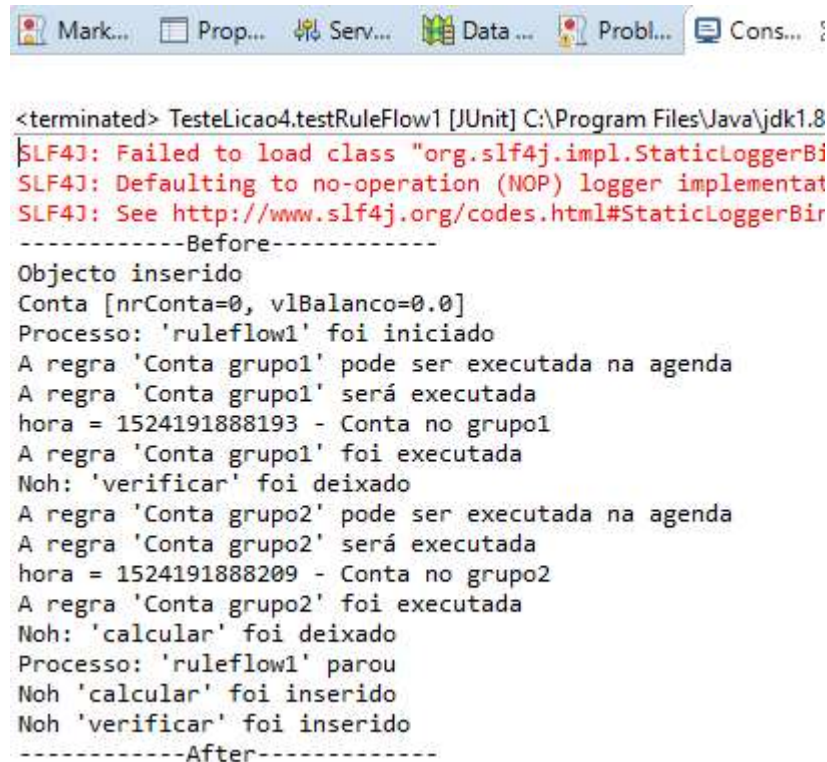
        sessionStatefull.fireAllRules();
    }
}

```


}

Antes de chamar o método `fireAllRules`, o método `startProcess` é chamado passando o parâmetro "RF1", que é o ID dado ao processo acima.

No console aparecerá:



```
<terminated> TesteLicao4.testRuleFlow1 [JUnit] C:\Program Files\Java\jdk1.8
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
-----Before-----
Objecto inserido
Conta [nrConta=0, vlBalanco=0.0]
Processo: 'ruleflow1' foi iniciado
A regra 'Conta grupo1' pode ser executada na agenda
A regra 'Conta grupo1' será executada
hora = 1524191888193 - Conta no grupo1
A regra 'Conta grupo1' foi executada
Noh: 'verificar' foi deixado
A regra 'Conta grupo2' pode ser executada na agenda
A regra 'Conta grupo2' será executada
hora = 1524191888209 - Conta no grupo2
A regra 'Conta grupo2' foi executada
Noh: 'calcular' foi deixado
Processo: 'ruleflow1' parou
Noh 'calcular' foi inserido
Noh 'verificar' foi inserido
-----After-----
```

Um `ruleflow-group` funciona como um grupo separado de regras. Aqueles que estão definindo o foco quando a etapa de regra é chamada com o mesmo ID do nó que o grupo do fluxo de regra. Quando as regras podem ser disparadas mais disparadas, o processo pode continuar no próximo nó.

1.6.2. Iniciando um fluxo de regras a partir de uma regra

Escreva a seguinte regra:

```
rule "start process"
when
then
    kcontext.getKieRuntime().startProcess("RF1");
end
```

Use o teste abaixo:

```
@Test
public void testRuleFlow2() {
    sessionStatefull = KnowledgeSessionHelper
        .getStatefulKnowledgeSessionForJBPM(kieContainer, "ksession-licao4");
```

```

OutputDisplay display = new OutputDisplay();
sessionStatefull.setGlobal("resultado", display);

```

```

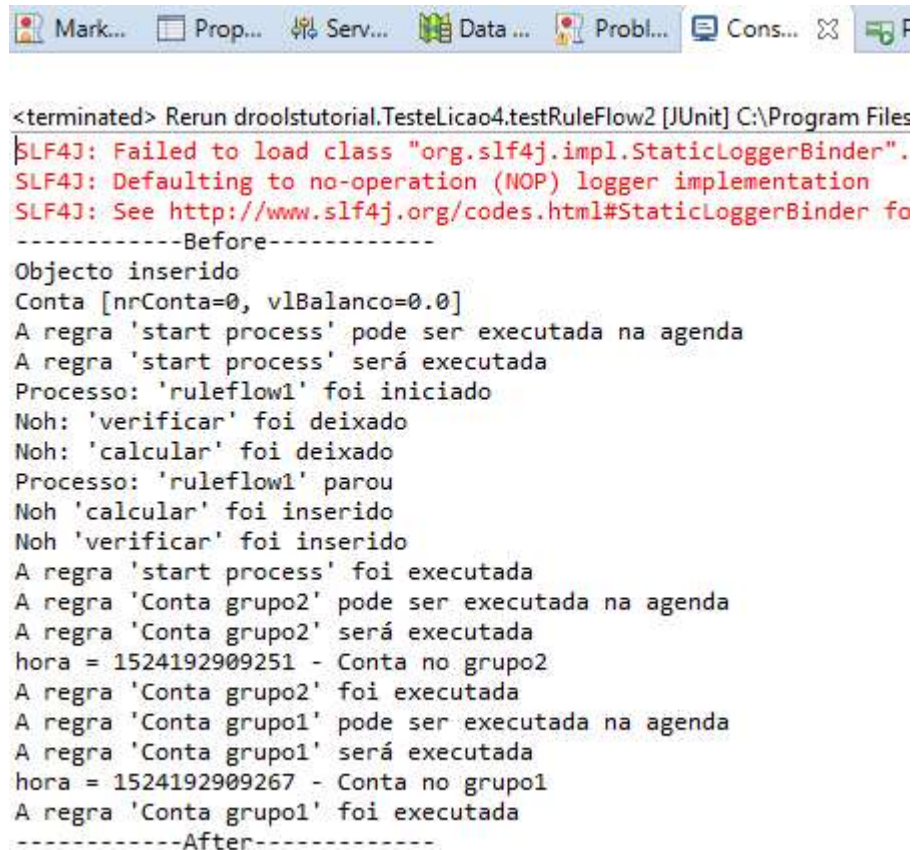
Conta conta = new Conta();
sessionStatefull.insert(conta);

```

```

sessionStatefull.fireAllRules();
}

```



```

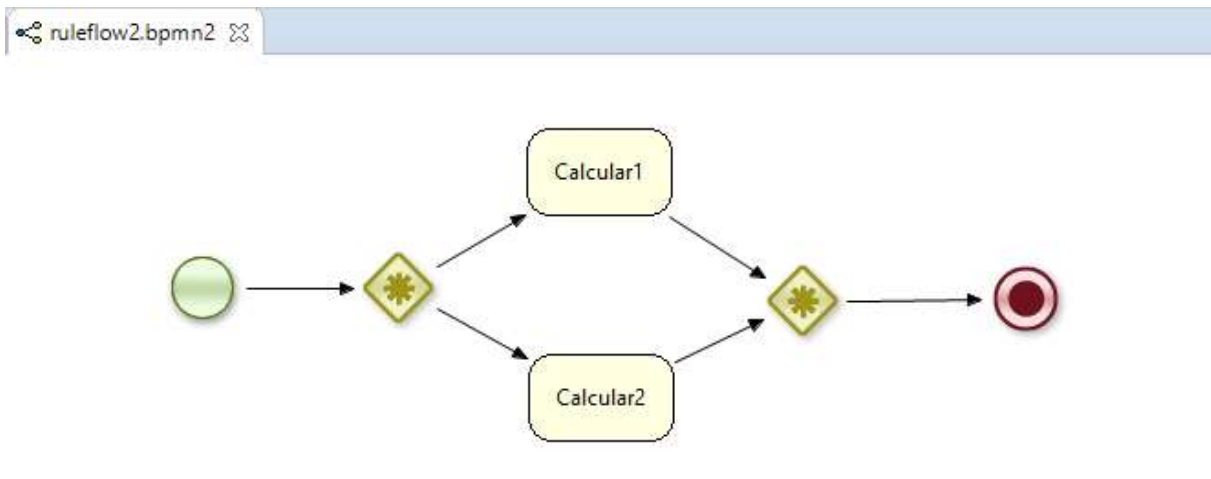
Mark... Prop... Serv... Data ... Probl... Cons...
<terminated> Rerun droolstutorial.TesteLicao4.testRuleFlow2 [JUnit] C:\Program Files
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder fo
-----Before-----
Objecto inserido
Conta [nrConta=0, vlBalanco=0.0]
A regra 'start process' pode ser executada na agenda
A regra 'start process' será executada
Processo: 'ruleflow1' foi iniciado
Noh: 'verificar' foi deixado
Noh: 'calcular' foi deixado
Processo: 'ruleflow1' parou
Noh 'calcular' foi inserido
Noh 'verificar' foi inserido
A regra 'start process' foi executada
A regra 'Conta grupo2' pode ser executada na agenda
A regra 'Conta grupo2' será executada
hora = 1524192909251 - Conta no grupo2
A regra 'Conta grupo2' foi executada
A regra 'Conta grupo1' pode ser executada na agenda
A regra 'Conta grupo1' será executada
hora = 1524192909267 - Conta no grupo1
A regra 'Conta grupo1' foi executada
-----After-----

```

1.6.3. Fluxo de regras com condições

Também é possível executar determinados grupos de regras com base na condição passada, que pode ter a mesma sintaxe que uma restrição de regra. Veja a seguir.

Crie um pacote chamado `licao4a`, um novo arquivo de processo chamado `ruleflow2.bpmn2` e um arquivo de regras `ruleflow2.drl`. O processo BPMN será similar ao apresentado abaixo e terá o ID "RF2":



A divisão à esquerda deve ser "Gateway divergente" e a da direita "Gateway de convergência". O Calcular1 deve ter um fluxo de regra chamado "grupo1" e o Calcular2 deverá ter o "grupo2".

Ao clicar no "diverge gateway" você deve selecionar o tipo "OR" e no "converge Gateway" o "XOR".

Property	Value
Constraints	
Id	4
MetaData	{Uniqueld=9f7feb87-d3bb-4970-8569-24e465463845, x=193, width=49, y=101, height=49}
Name	Gateway
Type	OR

Clique nos três pontos do valor da Constraint para editá-la. Você irá editar cada conexão para "To node Calcular1". Não esqueça de clicar no botão imports e importar a classe Conta().

Edit Constraints

To node Calcular1:

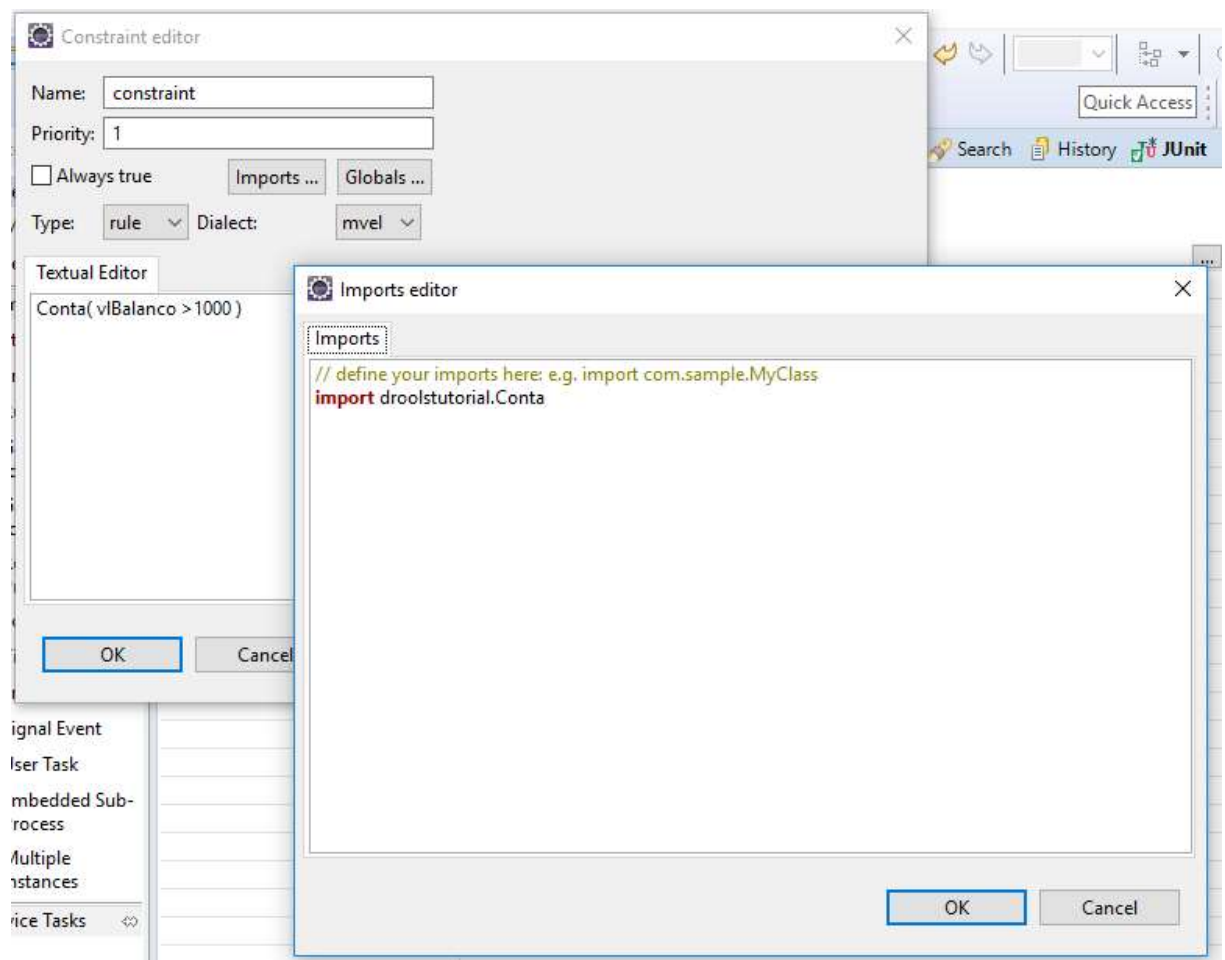
Edit

To node Calcular2:

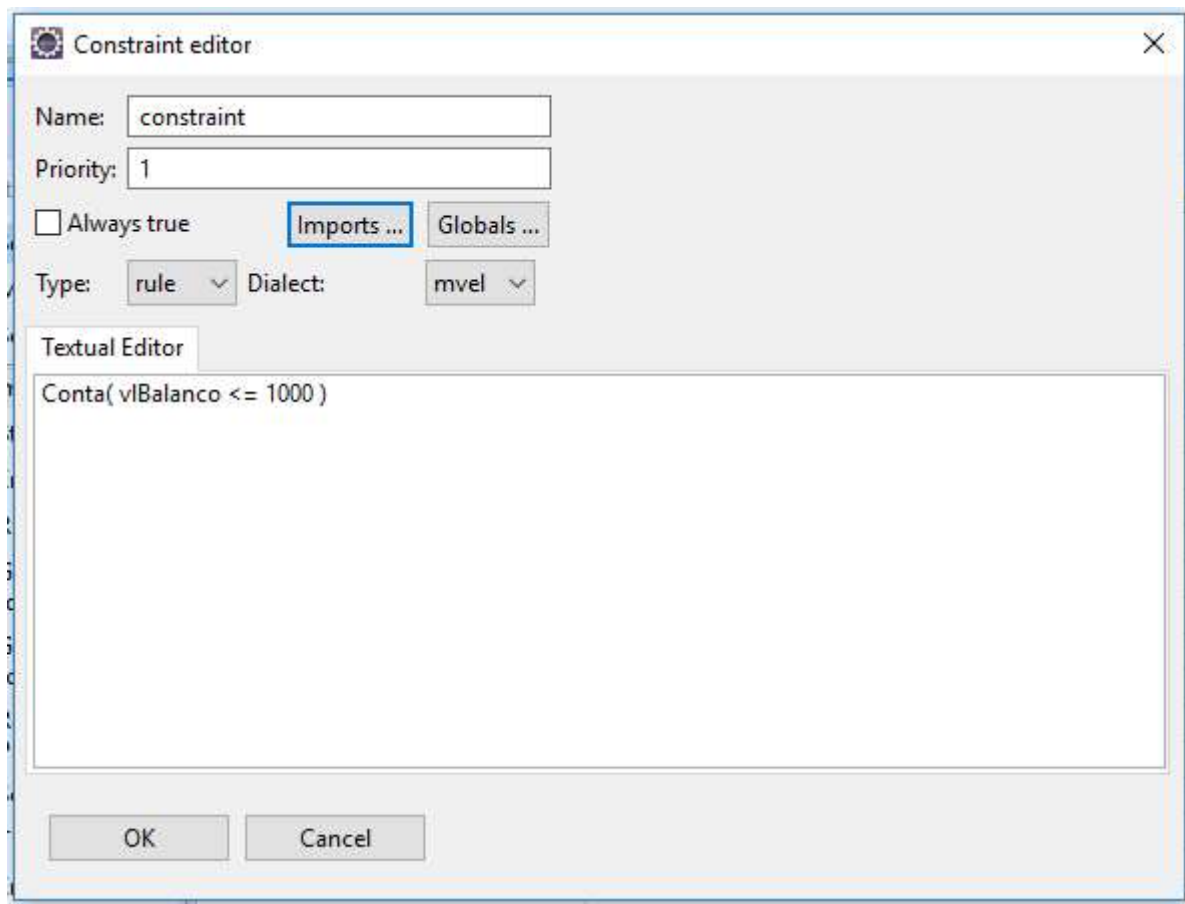
Edit

OK

Cancel



Faça o mesmo para o “To node Calcular2”:



Acrescente no kmodule.xml o trecho:

```
<kbase name="rules4a" packages="licao4a">
  <ksession name="ksession-licao4a"/>
</kbase>
```

E aqui está o arquivo de regras:

```
@Test
public void testRuleFlow3() {
    sessionStatefull = KnowledgeSessionHelper
        .getStatefulKnowledgeSessionForJBPM(kieContainer, "ksession-licao4a");

    OutputDisplay display = new OutputDisplay();
    sessionStatefull.setGlobal("resultado", display);

    Conta conta = new Conta();
    conta.setVIBalanco(2500);
    //conta.setVIBalanco(500);
    sessionStatefull.insert(conta);

    PeriodoContabil periodoContabil = new PeriodoContabil();
    sessionStatefull.insert(periodoContabil);

    sessionStatefull.fireAllRules();
}
```



```

Console
<terminated> Rerun droolstutorial.TesteLicao4.testRuleFlow3 [JUnit] C:\Program Files\Java\jdk1.8.0_152\
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
-----Before-----
Objecto inserido
Conta [nrConta=0, vlBalanco=2500.0]
Objecto inserido
PeriodoContabil [dtInicio=null, dtFim=null]
A regra 'RuleFlow-Split-RF2-4-2-DROOLS_DEFAULT' pode ser executada na agenda
A regra 'start process' pode ser executada na agenda
A regra 'start process' será executada
Processo: 'ruleflow2' foi iniciado
Noh: 'Calcular1' foi deixado
Processo: 'ruleflow2' parou
Noh 'Calcular1' foi inserido
A regra 'start process' foi executada
A regra 'Conta grupo1' pode ser executada na agenda
A regra 'Conta grupo1' será executada
hora = 1524195870990 - Conta no grupo1 > 1000
A regra 'Conta grupo1' foi executada
-----After-----

```

Se você mudar o vlBalanco para 500, então no console aparecerá:

```

Console
<terminated> Rerun droolstutorial.TesteLicao4.testRuleFlow3 [JUnit] C:\Program Files\Java\jdk1.8.0_152\
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
-----Before-----
Objecto inserido
Conta [nrConta=0, vlBalanco=500.0]
Objecto inserido
PeriodoContabil [dtInicio=null, dtFim=null]
A regra 'RuleFlow-Split-RF2-4-3-DROOLS_DEFAULT' pode ser executada na agenda
A regra 'start process' pode ser executada na agenda
A regra 'start process' será executada
Processo: 'ruleflow2' foi iniciado
Noh: 'Calcular2' foi deixado
Processo: 'ruleflow2' parou
Noh 'Calcular2' foi inserido
A regra 'start process' foi executada
A regra 'Conta grupo2' pode ser executada na agenda
A regra 'Conta grupo2' será executada
hora = 1524195977979 - Conta no grupo2 <= 1000
A regra 'Conta grupo2' foi executada
-----After-----

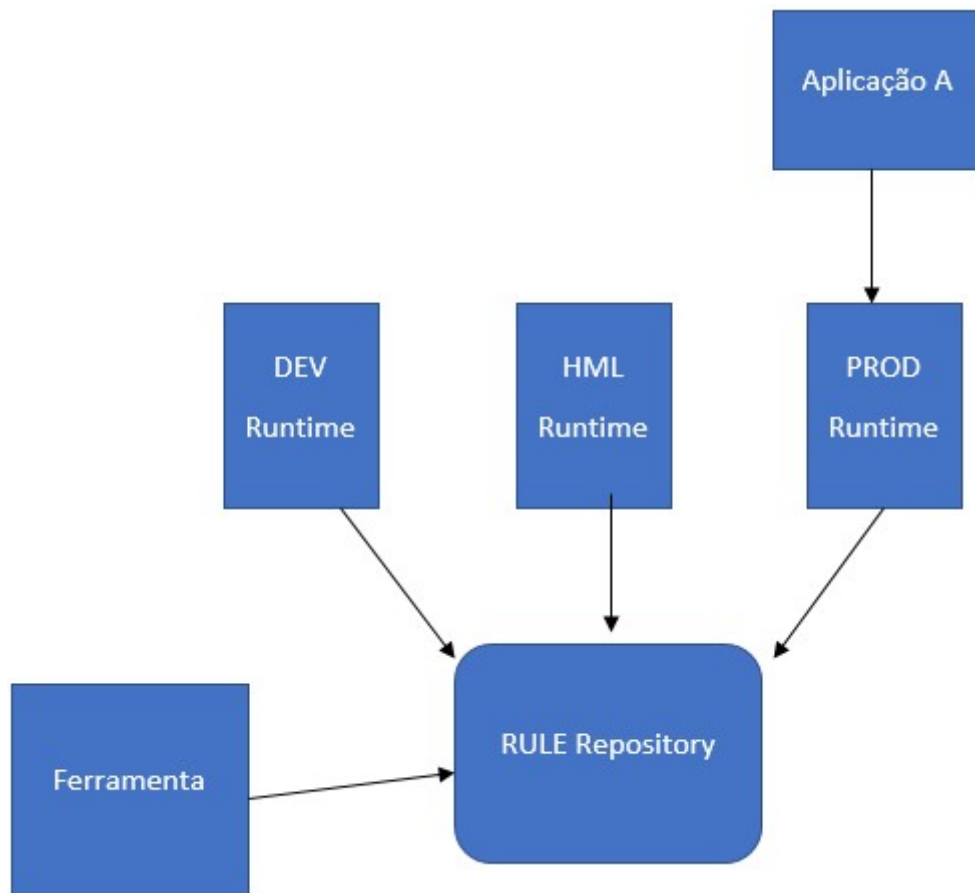
```

É mais eficiente ter dois grupos de regras como este, adicionando para todas as regras do "grupo1" a restrição no balanço > 1000 e o balanço <= 1000 para o "grupo2". De fato, se o nível R\$ 1.000,00 mudar, você terá que modificar todas as regras. E, além disso, se o usuário final der a sua regra: "o primeiro caso é quando o saldo é menor que R\$

1.000,00", a boa prática é implementar as regras de negócios conforme elas são fornecidas. E a implementação com um fluxo de regra ajudará os usuários finais a dividir sua maneira de expressar regras mais complexas.

2. Tutorial BRMS

Nas versões **5.x** do Drools, a arquitetura oferecida era similar a isto:



- Uma aplicação A chama um serviço Drools;
- O Drools Runtime é criado para o aplicativo usando a API drools padrão. Em muitos casos, o tempo de execução está usando uma sessão sem estado;
- O Drools Runtime está carregando o pacote de regras de um repositório de regras que possui uma ferramenta de autoria para implementar as regras. Estes dois últimos recursos foram implementados usando o que foi chamado de Guvnor.
- A implantação de uma nova versão de um pacote pode ser feita atualizando o Runtime remotamente chamando os recursos da API do Drools.

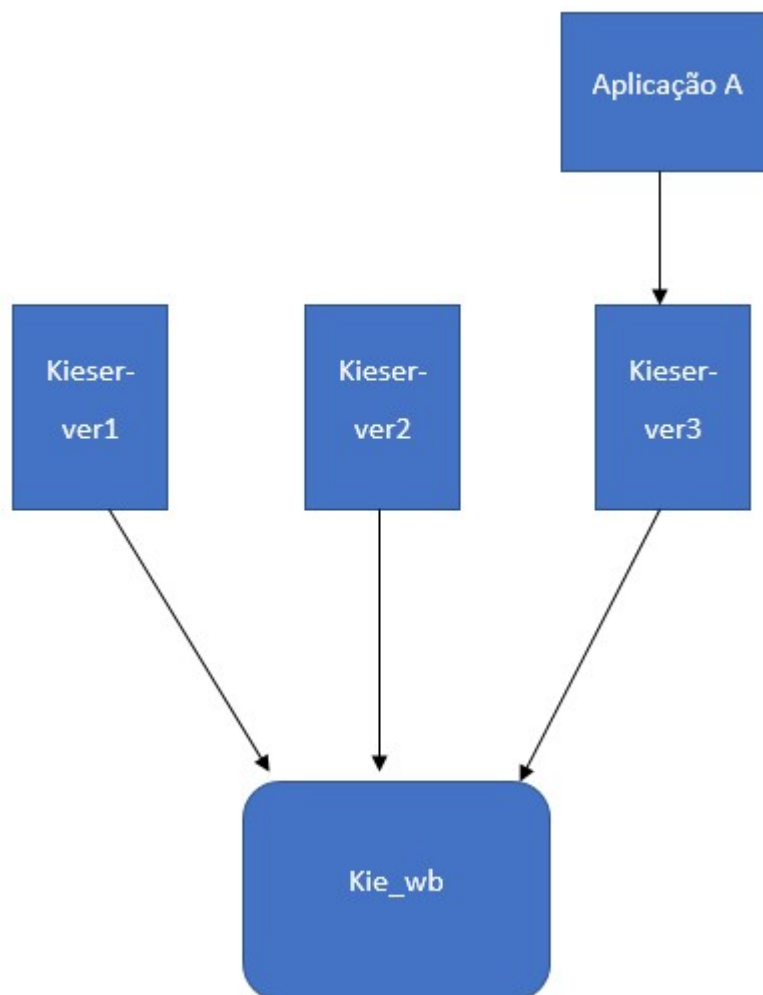
Uma alternativa desta arquitetura é implementar o Runtime diretamente no aplicativo

A.

Na série **6.x**, a ferramenta Guvnor foi substituída pelo que foi chamado de workbench e agora Business Central. Nos exemplos ele será chamado de *kie-wb*. É possível pegar a mesma arquitetura usando a API do drools e o kie-wb e existem agora duas possibilidades:

- Como o Business Central é um repositório maven, reconstrua o runtime (ou aplicativo se o runtime estiver incorporado) com a nova versão do pacote;
- No runtime, adicione dinamicamente ao caminho de classe java (do novo jar do pacote de regras) cada vez que houver uma nova versão.

Nesta versão, o novo componente introduzido, o kie-server, envolve tudo o que foi descrito na etapa anterior e é integrado ao ambiente de trabalho. A nova arquitetura se parece com a anterior:



- Cada kie-server chama o kie-wb na inicialização e se declara;
- A partir do kie-wb, é possível criar o kie-container no kie-server desejado;
- Este kie-container contém um artefato maven definido no kie-wb para uma versão;

- Tudo isso é vinculado usando a configuração do maven e todos os recursos relacionados. Você verá como fazer isso mais a frente;
- O kie-container expõe um serviço REST básico que tem a mesma assinatura da API que usada neste tutorial: inserir objeto, disparar todas as regras, etc.;
- O aplicativo A chama a API do drools remotamente e não precisa cuidar da implantação da nova versão da regra;

O toolkit do Drools, desde a versão 6.4, oferece uma ferramenta de autoria com Runtime pronto para trabalhar/implantar um conjunto de regras Drools em um pacote maven. A ferramenta de criação (kie-wb) ou o servidor de execução (kie-server) podem ser estendidos.

2.1.1. Instalando e configurando kie-wb e kie-server

Para rodar o Workbench e o Kie-server nativamente em sua máquina é necessário ter instalado o Java Runtime e, para facilitar este processo, use containers do Docker.

A tecnologia do Docker permite usar imagens pré-compiladas que são executadas em todas as plataformas suportadas por ele: quase todos os tipos de linux, Mac OS e windows, e isso no modo nativo para Windows ou MacOS. Você pode baixar o Docker e usar ou baixar a Toolbox dele.

O Docker Toolbox evita conflitos de sistemas operacionais, ao usar máquinas virtuais em seu host.

Baixe o Docker Toolbox para Windows clicando [aqui](https://docs.docker.com/toolbox/toolbox_install_windows/)³. Há também versões para Linux e Mac.

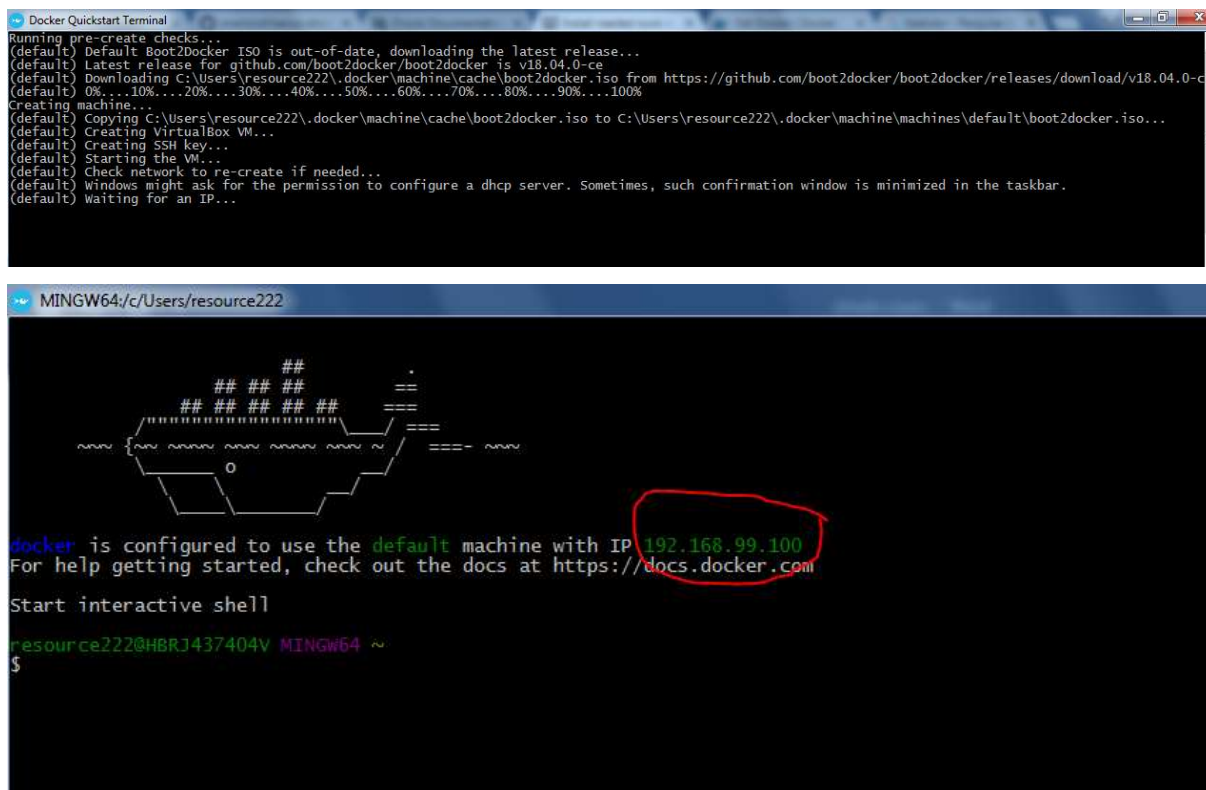
³ https://docs.docker.com/toolbox/toolbox_install_windows/



A instalação é simples. Faça o download, clique em install, next, next, ... finish e reinicie o pc.

As vezes acontece do virtualbox não subir, então verifique as configurações de virtualização da sua máquina na BIOS, ative-a, e reinstale o Docker Toolbox.

Uma vez instalado com sucesso, abra "Docker Quickstart Terminal" para que o Docker suba a máquina virtual e um IP será disponibilizado.



Agora você está pronto para começar o workbench!

Digite o comando abaixo sem as aspas para baixar a imagem e a executar:

“docker run -p 8081:8080 -p 8001:8001 -d --name drools-wb jboss/drools-workbench-showcase:7.7.0.Final”

Detalhes do comando:

- “-p” indica a porta que será utilizada, verifique se a porta 8081 está disponível para uso. Se não estiver, mude para outra que estiver disponível.
- “--name” é o nome do container que será criado;
- “jboss/drools-workbench-showcase:7.7.0.Final” é a versão que você está baixando. O “showcase” já vem com configurações default para usuário e senha, isso pode ser modificado posteriormente.

```
resource222@HBRJ437404V MINGW64 ~  
$ docker run -p 8081:8080 -p 8001:8001 -d --name drools-wb jboss/drools-workbench-showcase:7.6.0.Final  
Unable to find image 'jboss/drools-workbench-showcase:7.6.0.Final' locally  
7.6.0.Final: Pulling from jboss/drools-workbench-showcase  
469cfcc7a4b3: Pull complete  
b48bf37373e9: Pull complete  
0072cbc40985: Pull complete  
d00d92c6f1f0: Pull complete  
d769e58ab396: Pull complete  
ec5dd8cedf74: Pull complete  
ff7e687cf0b4: Pull complete  
4336beebcd88: Pull complete  
7421054213db: Pull complete  
567185a7d47c: Pull complete  
811ad2f06773: Pull complete  
e5e22a45971b: Pull complete  
Digest: sha256:d69e75f575be290b8da0f1ca1837cb29559ea743e0914d67cddc71bcc879014a  
Status: Downloaded newer image for jboss/drools-workbench-showcase:7.6.0.Final  
e88bcf18e6414efbd381b09aa8b7eba64bb9f2957dd2cebca110fcae76f38295
```

Para mais informações sobre a instalação do container do workbench, acesse este [link](#)⁴.

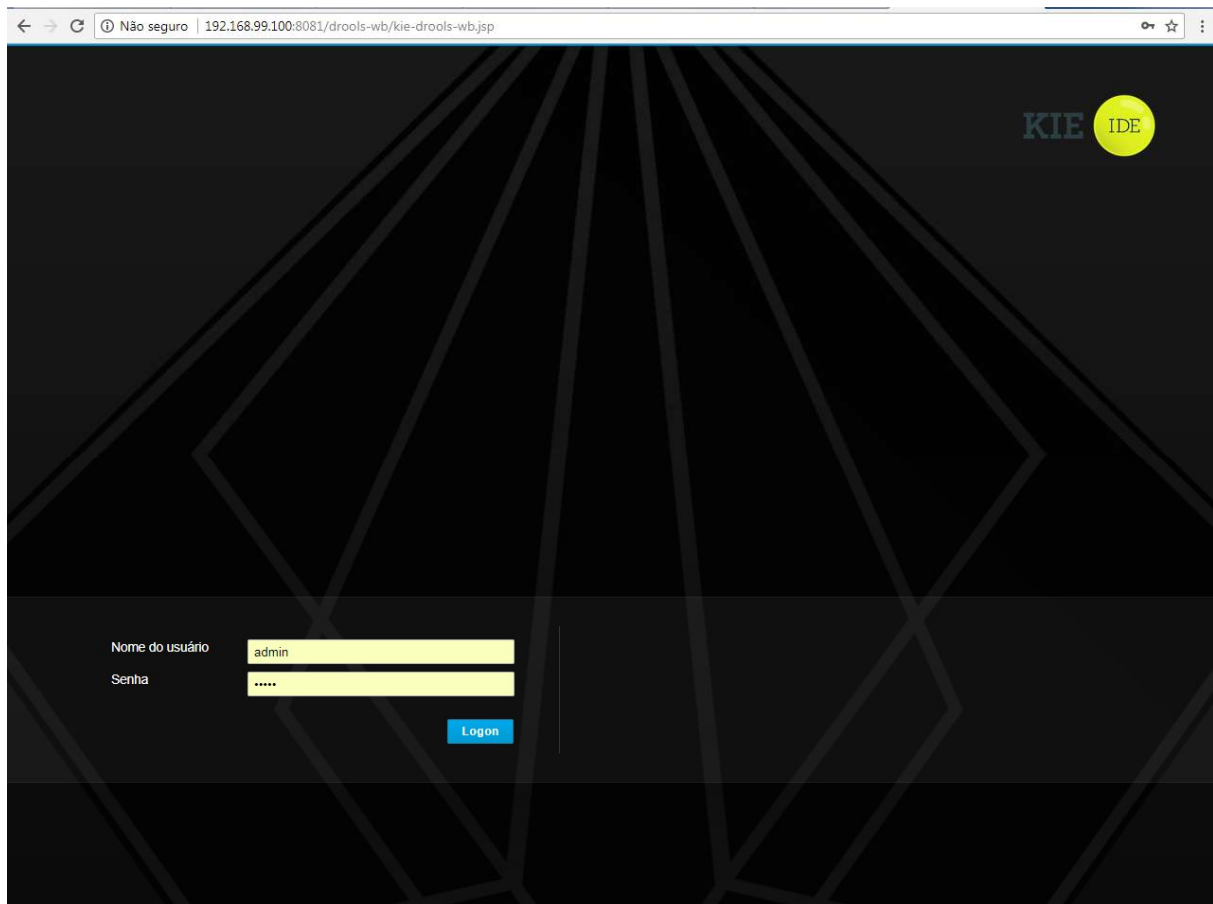
Para verificar se o container está no ar, execute o comando “docker ps”.

```
resource222@HBRJ437404V MINGW64 ~  
$ docker ps  
CONTAINER ID        IMAGE                                     COMMAND                  CREATED             STATUS              PORTS  
e88bcf18e641        jboss/drools-workbench-showcase:7.6.0.Final  "/start_drools-wb.sh"    2 minutes ago       Up 2 minutes       0.0.0.0:8001->8001/tcp, 0.0.0.0:8081->8081/tcp
```

Você agora pode acessar o workbench pelo endereço abaixo:

http://192.168.99.100:8081/drools-wb

⁴ <https://hub.docker.com/r/jboss/drools-workbench-showcase/>



Pronto! O usuário e senha padrão criado pela versão showcase, respectivamente, é admin e admin.

Agora instale o kie-server, o processo é similar ao do workbench. Digite o comando abaixo sem as aspas:

`"docker run -p 8082:8080 -d --name kie-server --link drools-wb:kie_wb jboss/kie-server-showcase:7.7.0.Final"`.

Detalhes do comando:

- A porta 8082 foi utilizada para não entrar em conflito com o workbench.
- "--link" vincula o drools workbench ao serviço do kie-server;
- "jboss/kie-server-showcase:7.7.0.Final" é a versão que você está baixando. O "showcase" já vem com configurações default para usuário e senha, assim como oworkbench, isso pode ser modificado posteriormente.

```
resource222@HBRJ437404V MINGW64 ~
$ docker run -p 8082:8080 -d --name kie-server --link drools-wb:kieserver jboss/kie-server-showcase:7.6.0.Final
Unable to find image 'jboss/kie-server-showcase:7.6.0.Final' locally
7.6.0.Final: Pulling from jboss/kie-server-showcase
469cfcc7a4b3: Already exists
b48bf37373e9: Already exists
0072cbc40985: Already exists
d00d92c6f1f0: Already exists
d769e58ab396: Already exists
8abb71d91b17: Pull complete
8b0d0398bdfd: Pull complete
a2d4cc10a859: Pull complete
77a470c262a0: Pull complete
94541247660c: Pull complete
710baf5006e: Pull complete
50cdf0fdcbec: Pull complete
e465950b8ba8: Pull complete
1d72c79818e3: Pull complete
19bb7c111b8c: Pull complete
f3649febc745: Pull complete
Digest: sha256:26f7b3aafc23c71523c53f049a33850bd58b7ef09f70e4baee9152373c023e13
Status: Downloaded newer image for jboss/kie-server-showcase:7.6.0.Final
e63f85e9abce3f483aa42a698e6c398882dee699f0a06a5503f5d19a5fa1baed
```

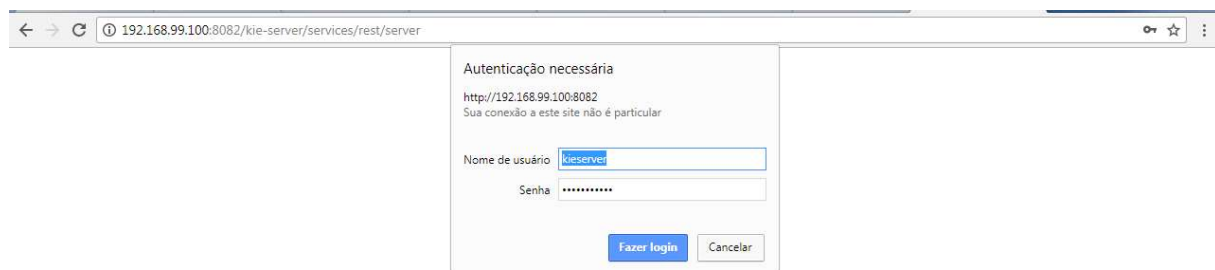
Para mais informações sobre a instalação do container do workbench, acesse este [link](#)⁵.

Digitando “docker ps” você verá o workbench e o kie-server rodando.

```
resource222@HBRJ437404V MINGW64 ~
$ docker ps
CONTAINER ID        IMAGE                                     COMMAND                  CREATED             STATUS              PORTS
e63f85e9abce        jboss/kie-server-showcase:7.6.0.Final  "/start_kie-server."    4 minutes ago       Up 4 minutes       0.0.0.0:8082->8080/tcp
e88bcf18e641        jboss/drools-workbench-showcase:7.6.0.Final  "/start_drools-wb.sh"  20 minutes ago      Up 20 minutes      0.0.0.0:8001->8001/tcp, 0.0.0.0:8002->8002/tcp
```

Pronto! Você pode verificar se o servidor está no ar digitando o endereço abaixo:

<http://192.168.99.100:8082/kie-server/services/rest/server>



O usuário e senha padrão criado pela versão showcase, respectivamente, é “kieserver” e “kieserver1!”.

⁵ <https://hub.docker.com/r/jboss/kie-server-showcase/>



```
<?xml version='1.0' encoding='UTF-8'>
<response type="SUCCESS" msg="Kie Server Info">
  <kie-server-info>
    <capabilities>KieServer</capabilities>
    <capabilities>BRM</capabilities>
    <capabilities>BPM</capabilities>
    <capabilities>CaseMgmt</capabilities>
    <capabilities>BPM-UI</capabilities>
    <capabilities>BRP</capabilities>
    <capabilities>DMN</capabilities>
    <capabilities>Swagger</capabilities>
  </kie-server-info>
  <location>
    http://172.17.0.3:8080/kie-server/services/rest/server
  </location>
  <messages>
    <content>
      Server KieServerInfo{serverId='kie-server-e63f85e9abce', version='7.6.0.Final', location='http://172.17.0.3:8080/kie-server/services/rest/server', capabilities=[KieServer, BRM, BPM, CaseMgmt, BPM-UI, BRP, DMN, Swagger]}started successfully at Fri Apr 20 15:46:23 UTC 2018
    </content>
    <severity>INFO</severity>
    <timestamp>2018-04-20T15:46:23.488Z</timestamp>
  </messages>
  <name>kie-server-e63f85e9abce</name>
  <id>kie-server-e63f85e9abce</id>
  <version>7.6.0.Final</version>
</kie-server-info>
</response>
```

Logo mais, você entenderá como o workbench e o kie-server trabalham juntos.

2.1.2. Business Central para Kie Drools Workbench

Na parte anterior, você aprendeu os conceitos básicos do Drools. Fez isso usando a IDE do eclipse para desenvolvimento Java. Porém não tem como esperar que um Usuário de Negócios use o eclipse como uma interface de usuário para implementar regras. Desde a versão 5 do drools, existe uma interface de usuário dedicada para isso que foi chamada Guvnor, chamada kie Workbench nas versões 6.x até 6.3 é chamada de Business central iniciando com a versão 6.4. Historicamente, o Guvnor é um BRMS = Business Rule Management System. Ele permitiu lidar com toda a vida útil de uma regra:

- CRUD em um artefato de regra,
- Padrão de regras avançadas, como tabelas de decisão,
- Definição do processo RuleFlow/jBPMN,
- Definição de pacotes de regras de implantação, etc.

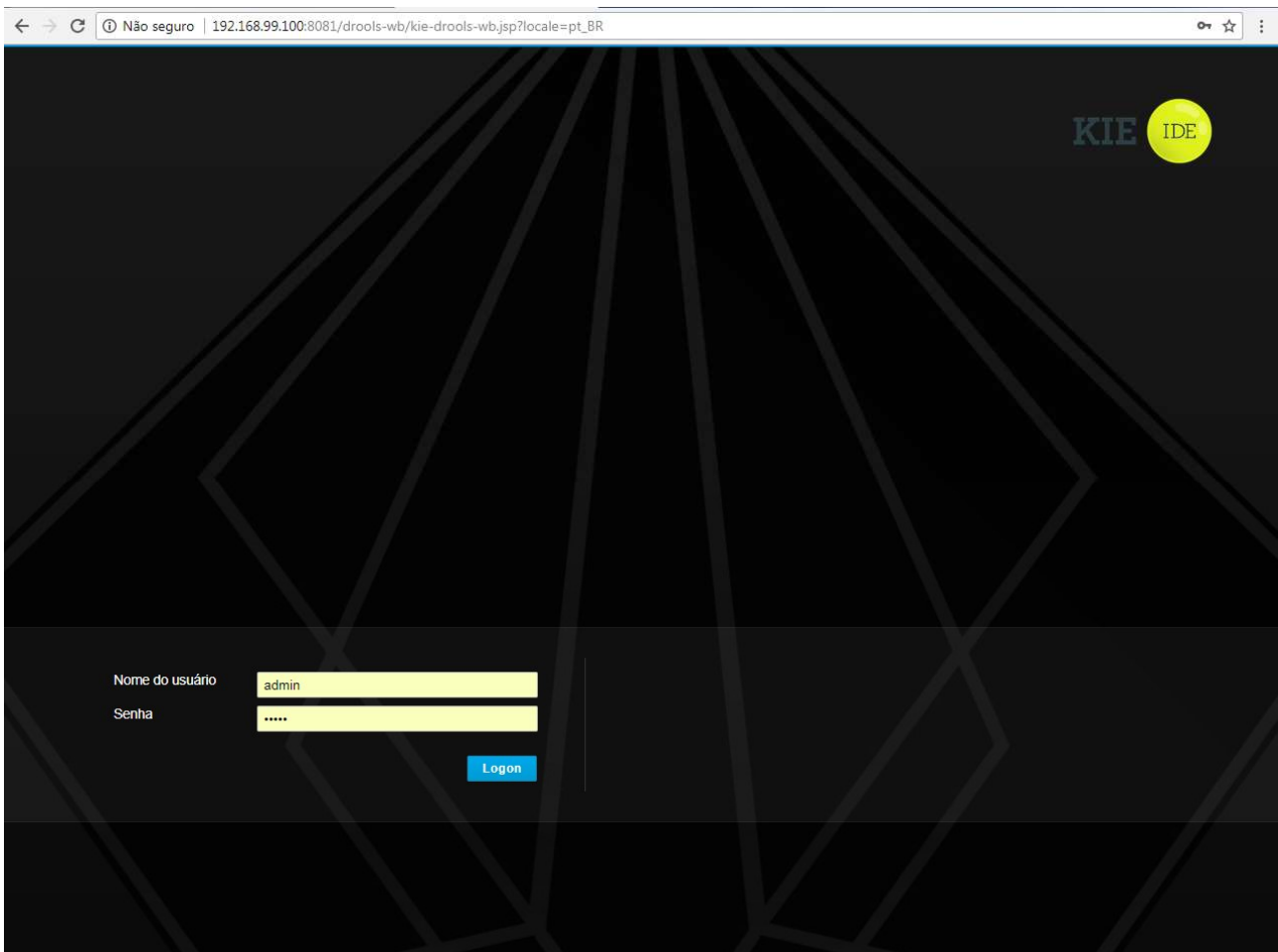
Tudo foi armazenado em um componente CMS (Content Management System) da fundação Apache Jackrabbit. O aplicativo Guvnor em si era uma aplicação monolítica cujos recursos eram apenas extensíveis chamando sua API de REST.


Da versão 7.x em diante ela é chamada de Kie Drools Workbench, o Drools tem suporte completo ao Runtime para DMN (Decision Model and Notation). Os arquivos DMN são agora um ativo que pode ser adicionado a qualquer kjar para execução. Consulte a seção DMN, da documentação oficial (Drools), para detalhes sobre como construir e executar modelos DMN.

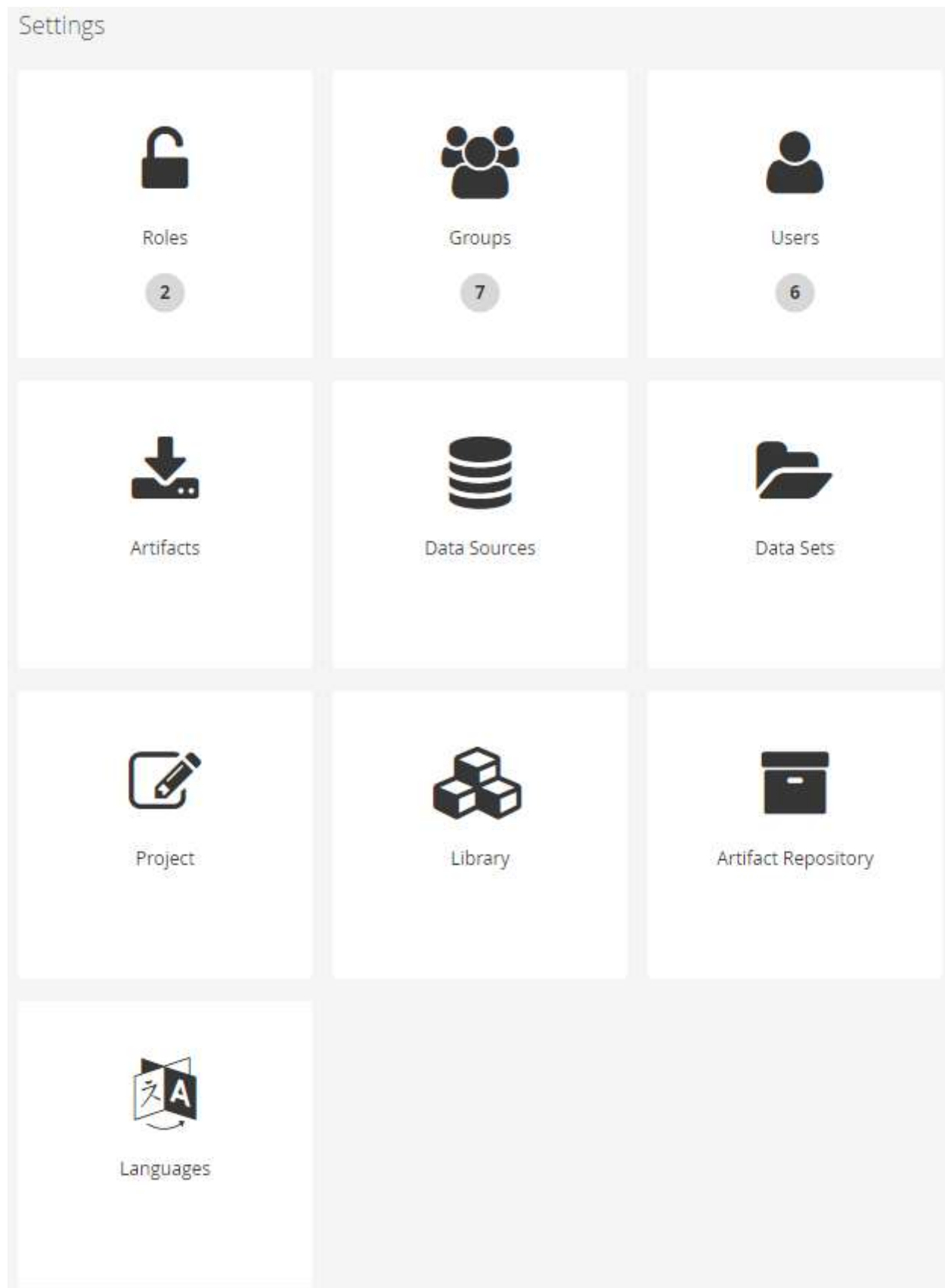
1.1.1. Kie Drools Workbench

Para facilitar o entendimento da ferramenta, você irá reproduzir o mesmo código criado no Caso de Teste:

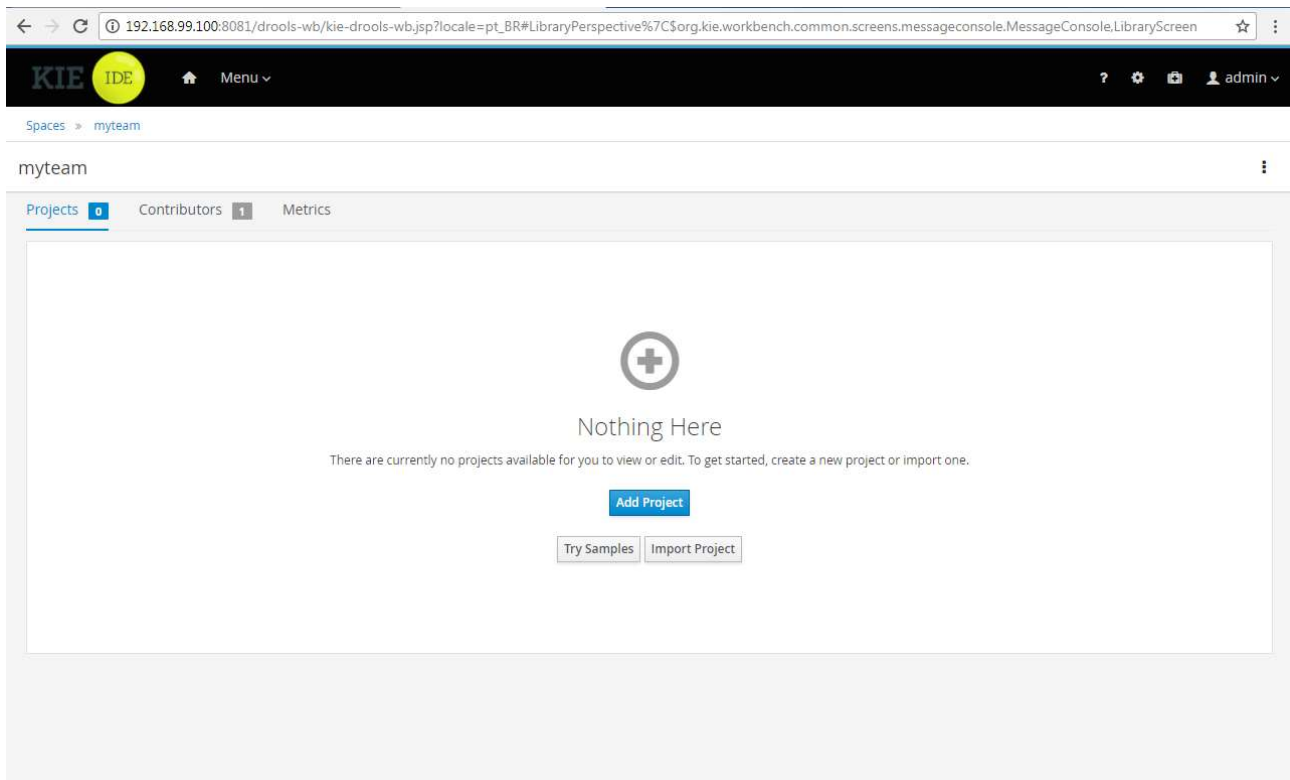
- Acesse o workbench pelo link <http://192.168.99.100:8081/drools-wb>. O usuário e a senha são, respectivamente: admin e admin.



- Clicando na catraca  que está no canto superior direito da tela, você terá acesso à tela de configurações. Nela, você determinará uma série de configurações que vão desde permissão de usuário a configuração do projeto e repositório do projeto. Explore-o depois.



- Volte para a tela inicial. Ao logar, clique em projetos (pode ser pelo botão do dashboard inicial ou pelo menu > projects).



- Clique em “Add Project”, acrescente as informações da imagem abaixo e clique em add. Um projeto com uma estrutura Maven será criado para você. O projecto criado aqui poder ser importado direto do GitHub⁶, mas o ideal é que você desenvolva direto do seu workbench.

⁶ <https://github.com/jmarlonsf/banco-regras-droolstutorial>

Add Project

Name

banco-regras

Description

Hide Advanced Options

Group ID

drools tutorial

Artifact ID

banco.regras

Version

1.0.0-SNAPSHOT

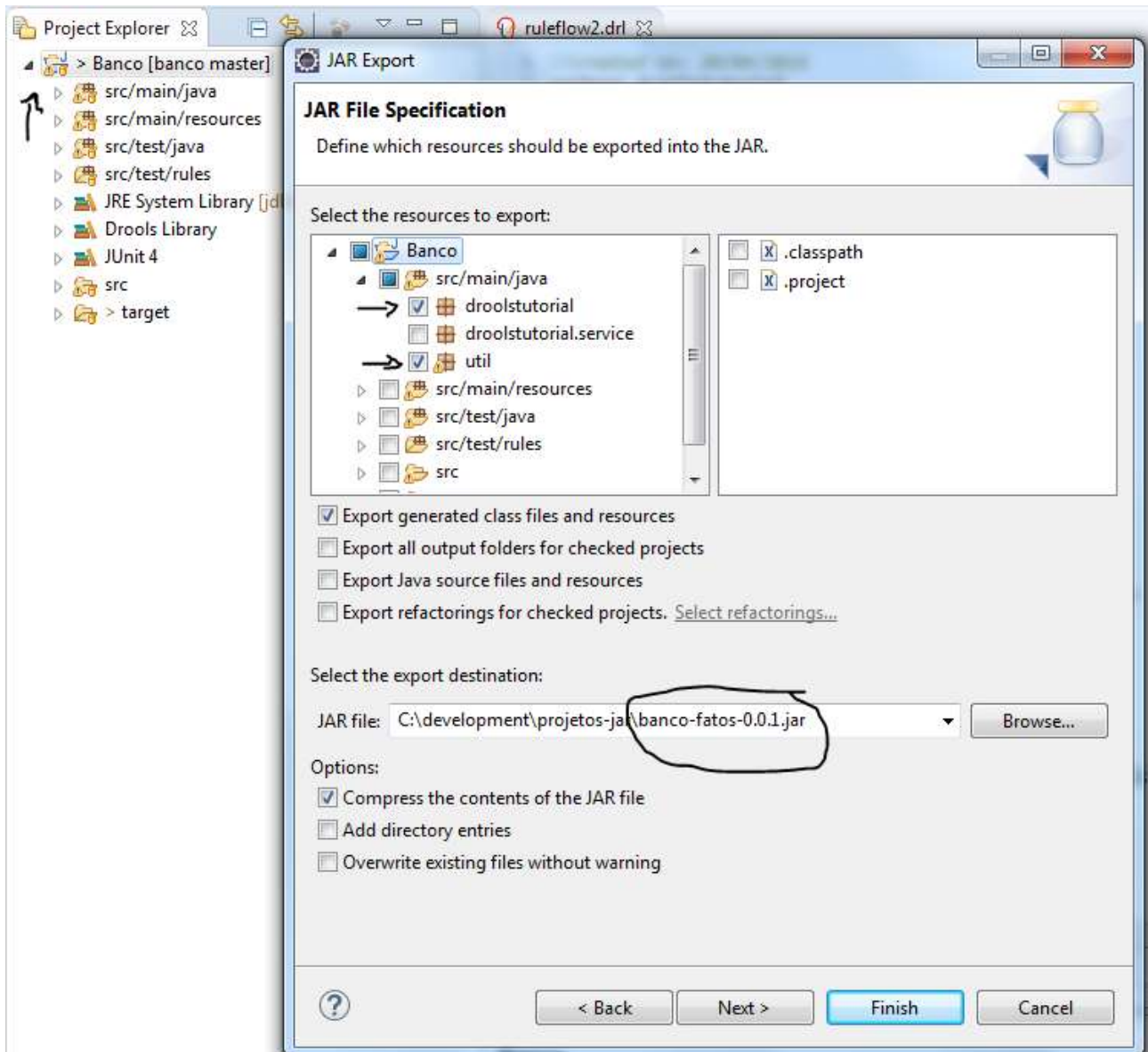
Cancel


Add

1.1.1.1. Importando os fatos

No eclipse, os fatos e as regras estavam integrados no mesmo projeto e, por isso, a comunicação entre eles era mais simples. Aqui no workbench é necessário exportar os fatos num .jar e importa-los dentro do projeto no workbench como dependência.

- Selecione o projeto no eclipse e exporte apenas as entidades e o pacote “útil” que foram criadas nele com o nome banco-fatos-1.0.0-SNAPSHOT.



- Volte à tela de configurações  e clique em Artifacts:



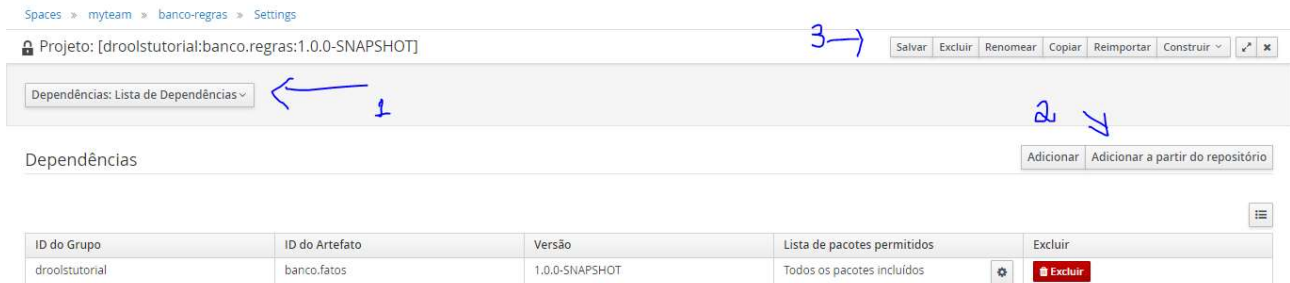
Artifacts

- Clique em carregar e, em selecionar arquivo, busque o banco-fatos-1.0.0-SNAPSHOT.jar que você criou e clique em carregar. Como o projeto de fatos não foi convertido para um projeto Maven, então ele irá reclamar que “o JAR não contém um POM válido”, clique em Ok e acrescente manualmente.



Pronto! O projeto de fatos já está dentro do repositório do Workbench. Agora você irá acrescentá-lo como dependência do seu projeto de fatos.

- Clique em Menu > Projects > banco-regras > Settings > No combo esquerdo selecione “Dependências” > Clique em “Adicionar a partir do repositório” > Selecione o .jar importado e clique em Save.



Seu projeto agora possui uma referência dos fatos que irá utilizar para criar os ativos de fluxo, regra etc.

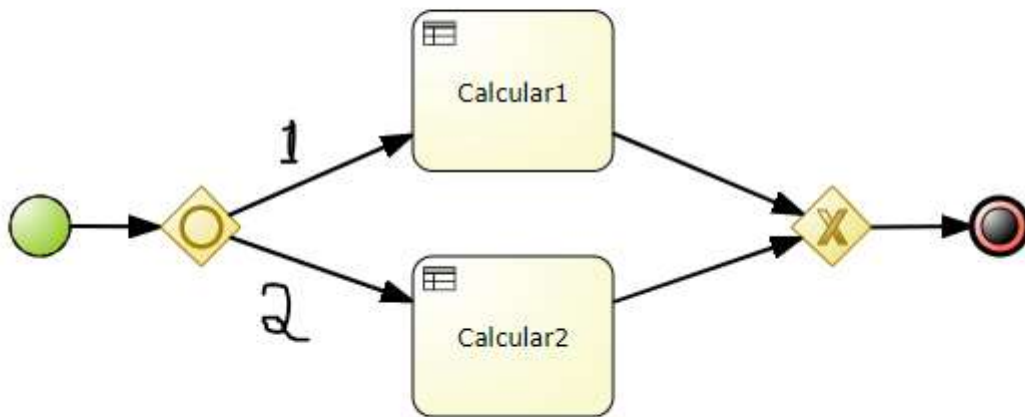
1.1.1.2. Criando fluxo de regras

Agora você irá reproduzir o segundo fluxo, o que foi criado no eclipse dentro do workbench. Não é obrigatório utilizar o mesmo nome, mas coloque-o só para facilitar o entendimento.

- Clique Add New Asset > Processo Comercial. Clique Ok.

A screenshot of a dialog box titled 'Criar Novo Processo Comercial'. It has a close button (X) in the top right corner. The dialog contains two input fields: 'Processo Comercial*' with the text 'ruleflow2' and 'Pacote' with a dropdown menu showing 'droolstutorial.banco.regras'. At the bottom right, there are two buttons: '+ Ok' and 'Cancelar'.

- Crie um projeto como o abaixo:



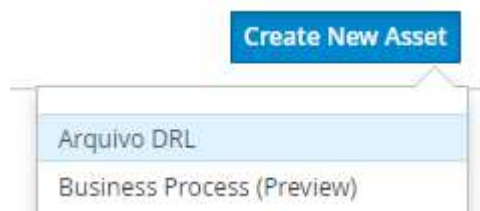
- Você consegue alterar as propriedades dos elementos ao clicar sobre eles.
 - Clique no background branco abra as propriedades do diagrama
 - Globais:
 - Nome: Conta;
 - Defined Types: Conta [droolstutorials];
 - ID: RF2;
 - Pacote: droolstutorial.banco.regras;
 - O Primeiro gateway é inclusivo;
 - A seta é quem recebe a condição.
 - Na seta 1:
 - Condição do Idioma de Expressão: drools.
 - Expressão: Conta(vlBanco > 1000);
 - Na seta 2:
 - Condição do Idioma de Expressão: drools.
 - Expressão: Conta(vlBanco <= 1000);
 - A tarefa precisa receber um nome, um grupo e um tipo.
 - Tarefa A:
 - Nome: Calcular1;

- Tipo: Tarefa de Regra Comercial;
- Grupo: grupo1.
- Tarefa B:
 - Nome: Calcular2;
 - Tipo: Tarefa de Regra Comercial;
 - Grupo: grupo2.

1.1.1.3. Criando arquivo de regras para o fluxo

O arquivo de regras seguirá o seguinte cenário:

- ♦ Regra 1: Quando for maior que 1000, atualize o número da conta para 1;
- ♦ Regra 2: Quando for menor ou igual a 1000, atualize o número da conta para 2.
- Clique Add New Asset > Arquivo DRL. Clique Ok.



 A screenshot of a dialog box titled 'Criar Novo Arquivo DRL'. It contains the following fields and options:

- A text field labeled 'Arquivo DRL *' with the value 'ruleflow2' entered.
- A dropdown menu labeled 'Pacote' with the value 'droolstutorial.banco.regras' selected.
- A checkbox labeled 'Usar o Domain Specific Language (DSL - Idioma Específico do Domain)' which is currently unchecked.
- At the bottom right, there are two buttons: '+ Ok' (blue) and 'Cancelar' (gray).

```
package droolstutorial.banco.regras;

//list any import classes here.
import droolstutorial.Conta;

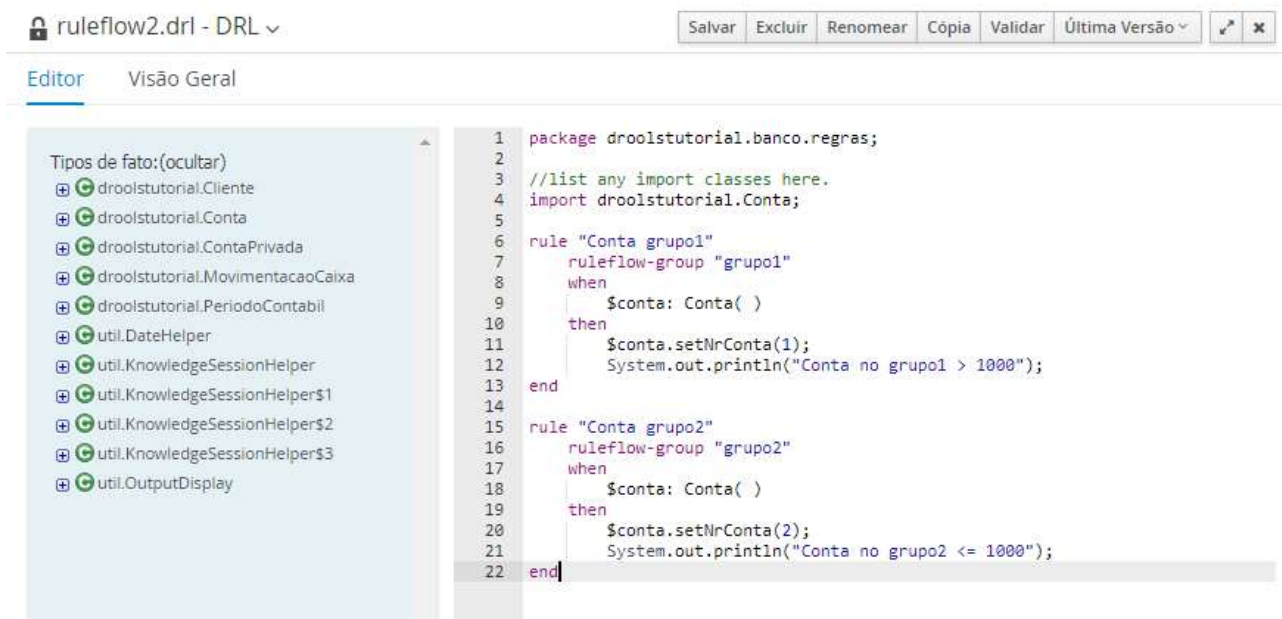
rule "Conta grupo1"
```

```

ruleflow-group "grupo1"
when
    $conta: Conta( )
then
    System.out.println("Conta no grupo1 > 1000");
    $conta.setNrConta(1);
end

rule "Conta grupo2"
ruleflow-group "grupo2"
when
    $conta: Conta( )
then
    System.out.println("Conta no grupo2 <= 1000");
    $conta.setNrConta(2);
end

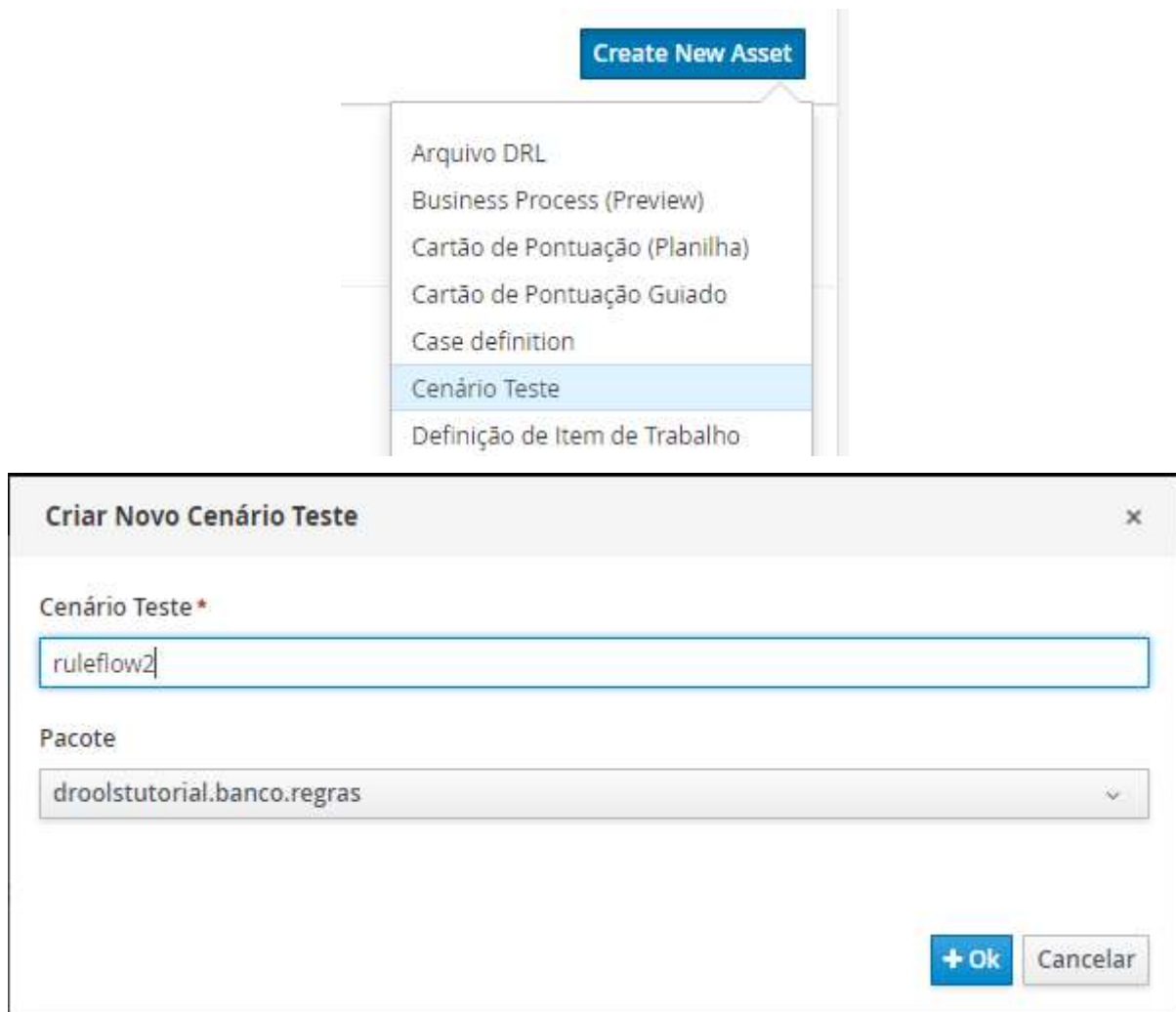
```



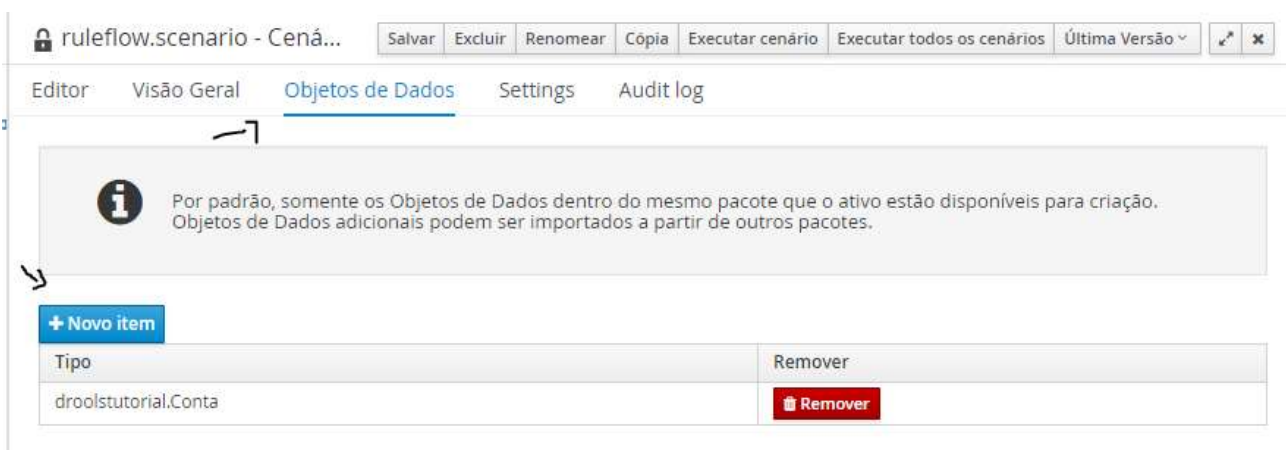
1.1.1.4. Criando cenário de testes

O teste no workbench é um pouco diferente do teste unitário do JUnit, mas é fácil de entender.

- Clique Add New Asset > Cenário Teste. Clique Ok.



- Em “Objetos de Dados”, clique em “+ Novo item” > importe droolstutorial.Conta > clique Ok.



- Volte ao Editor. Execute os seguintes comandos:
 - + Gerado > Inserir novo fato: Conta;
 - Nome do fato: conta1 > Adicionar;
 - Adicionar campo > nrConta > Ok;

- Editar > valor literal > 0;
 - Inserir 'Conta' > vlBalanco > ok;
 - Editar > valor literal > 2000.
- + Gerado > Ativar a regra do grupo de fluxo: grupo1 > Adicionar
- + Esperar > Regras: Conta grupo1 > Ok;
 - Disparou pelo menos uma vez;
- + Esperar > Regras: Conta grupo2 > Ok;
 - Não Disparou;
- + Esperar > Valor do fato: conta1 > Adicionar;
 - Conta 'conta1' possui valores > nrConta > Ok
 - Igual a > 1;
 - Conta 'conta1' possui valores > vlBalanco > Ok;
 - Igual a > 2000
- O arquivo ficará assim:

ruleflow2.scenario - Cen... Salvar Excluir Renomear Cópia Executar cenário

Editor Visão Geral Objetos de Dados Settings Audit log

+ GERADO

Inserir 'Conta' [conta1] ✕ 'Conta' facts

nrConta: ✕

vlBanco: ✕

✕

Ativar a regra do grupo de fluxo

grupo1 ✕

+ MÉTODO DE CHAMADA

Adicionar dados de entrada e expectativas aqui.

+ ESPERAR

Regras esperadas

Conta grupo1: ✕

Conta grupo2: ✕

Conta 'conta1' possui os valores:

nrConta: ✕ ✕ 'conta1'

vlBanco: ✕


Delete one scenario block above

- Clique em “Executar cenário”. Se todos os itens forem aprovados, então o teste executou com sucesso:

Audit log: dados de entrada e expectativas aqui.

+ ESPERAR


Regras esperadas

 Conta grupo1: ✕

 Conta grupo2: ✕

Conta 'conta1' possui os valores:

 nrConta: ✕ ✕ 'conta1'

 vlBanco: ✕

Delete one scenario block above



- Clicando em “Mais..” você pode continuar escrevendo outros cenários de testes.
- Crie outro teste abaixo com o seguinte cenário: Crie uma conta2 (nrConta=0, vlBalanço=500), ativando as regras do grupo2 o resultado será (nrConta=2, vlBalanço=500), executando pelo menos uma vez “Conta grupo2” e não executando “Conta grupo1”. O resultado será isso:

+ GERADO

Inserir 'Conta' [conta2]

nrConta: 0

vlBalanço: 500

'Conta' facts

Ativar a regra do grupo de fluxo

grupo2

+ MÉTODO DE CHAMADA

Adicionar dados de entrada e expectativas aqui.

+ ESPERAR

Regras esperadas

Conta grupo1: Não disparou

Conta grupo2: disparado pelo menos uma vez

Conta 'conta2' possui os valores:

nrConta: iguala 2

vlBalanço: iguala 500

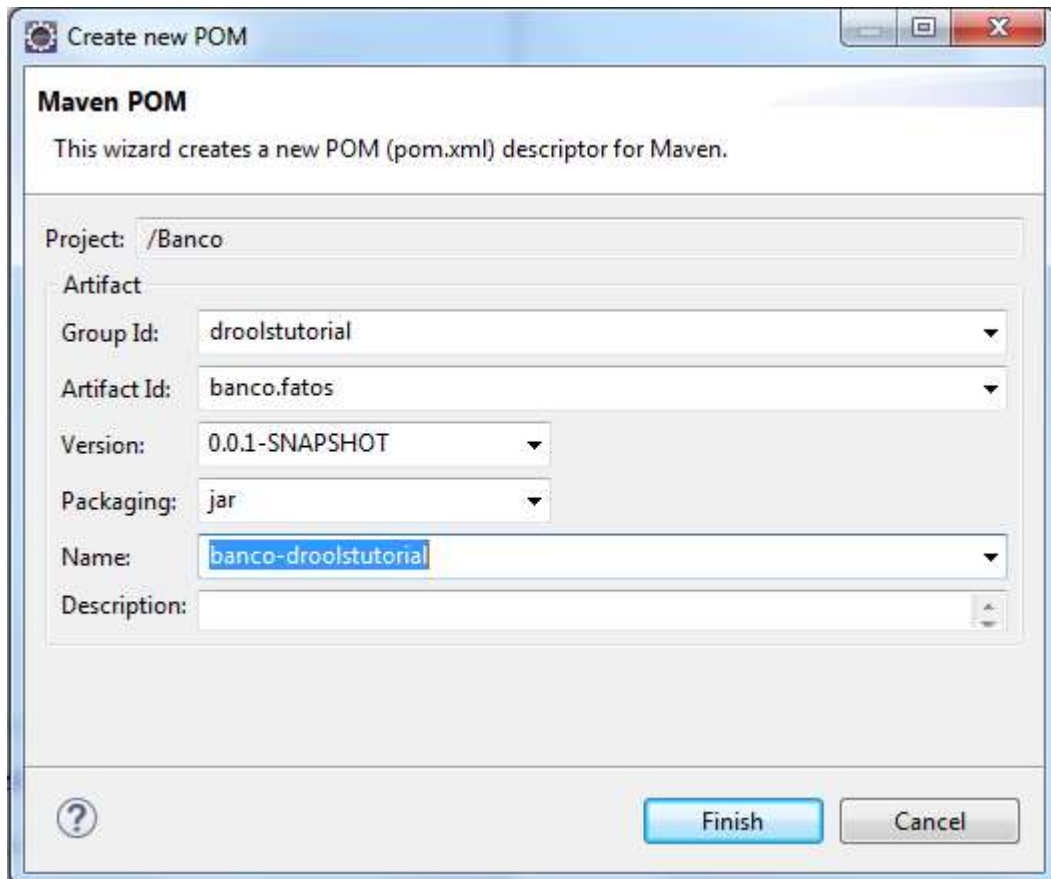
'conta2'

Delete one scenario block above

1.1.1.5. Executando regra do WB pelo Eclipse;

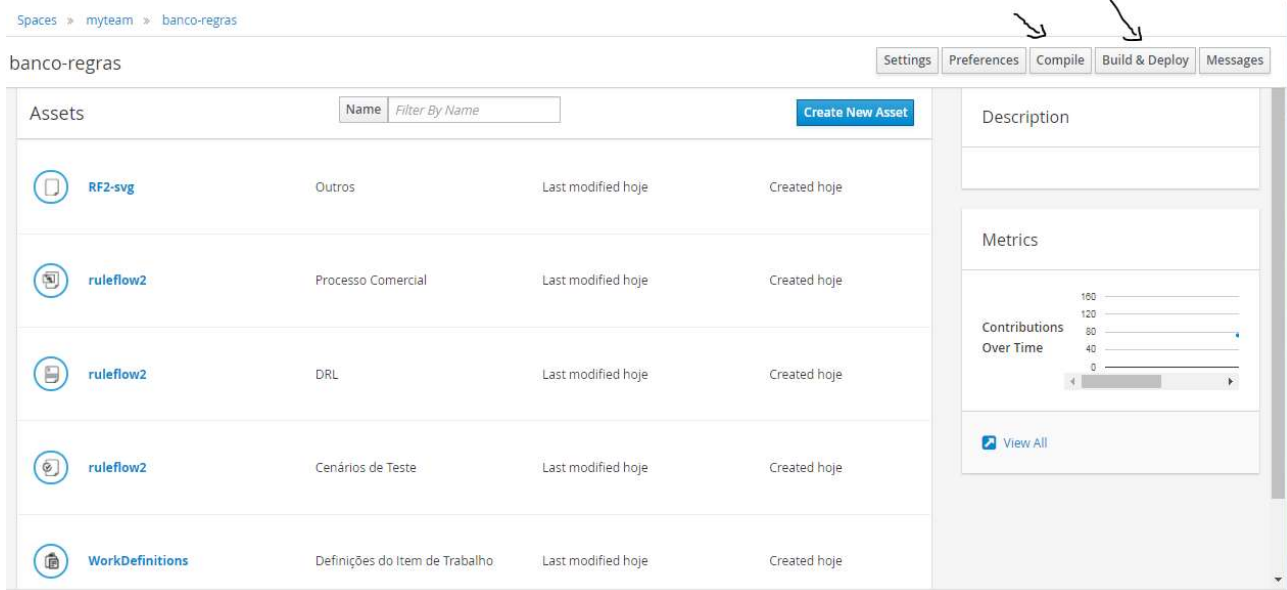
Agora chegou a hora da verdade. O acesso a regra é feito pelo kie-server⁷. A aplicação passa o fato para o servidor, que executa a regra exposta pelo container e retorna o fato validado.

- Configure o projeto para um “Maven Project”
 - ◆ Botão direito no projeto > Configure > Convert to Maven Project.

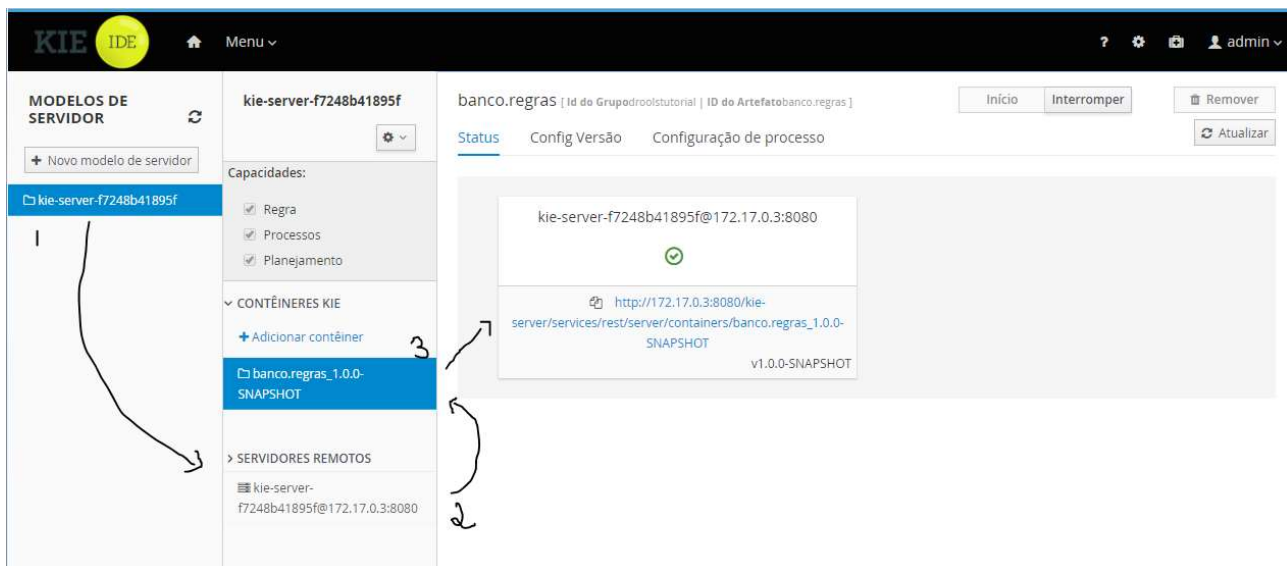


- Volte para a tela inicial do projeto > Clique em “Compile” > clique em “Build & Deploy”. Um container com a versão do projeto será exposto pelo kie-server.

⁷ <http://192.168.99.100:8082/kie-server/services/rest/server>



- Para visualizar o container vá em: Menu > Execution Servers.



- Para visualizar o container exposto, acesse a seguinte URL:

http://192.168.99.100:8082/kie-server/services/rest/server/containers/banco.regbras_1.0.0-SNAPSHOT

O eclipse vai bater neste container para executar a regra. Para gerar um novo container é necessário alterar a versão do projeto de regras, salvar, compilar e buildar novamente, e a nova versão estará disponível.

- Crie uma classe chamada BusinessRules, no pacote droolstutorial do source folder src/test/java

```
package droolstutorial;
```

```
import java.util.ArrayList;
```

```

import java.util.HashMap;
import java.util.HashSet;
import java.util.Set;

import org.kie.api.KieServices;
import org.kie.api.command.Command;
import org.kie.api.command.KieCommands;
import org.kie.api.runtime.ExecutionResults;
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.RuleServicesClient;

public class BusinessRules {

    //Servidor disponibilizado ao subir o container do Kie-server vinculado ao work-
    bench
    private static final String URL = "http://192.168.99.100:8082/kie-server/servi-
    ces/rest/server";
    //usuário e senha do kie-server
    private static final String USER = "kieserver";
    private static final String PASSWORD = "kieserver1!";
    //container exposto ao buildar o projeto de regras
    private static String CON-
    TAINER_ID = "banco.regras_1.0.0-SNAPSHOT";

    private static final MarshallingFormat FORMAT = MarshallingFormat.XSTREAM;

    private KieServicesConfiguration conf;
    private KieServicesClient kieServicesClient;

    public void initialize(HashMap<String, Object> fatos) {
        conf = KieServicesFactory.newRestConfiguration(URL, USER, PASSWORD);

        Set<Class<?>> extraClass = new HashSet<Class<?>>();

        fatos.forEach((id, fato) -> extraClass.add(fato.getClass()));

        //Adiciona as classes/fatos enviados à chamada
        conf.addExtraClasses(extraClass);

        conf.setUseSsl(true);

        conf.setMarshallingFormat(FORMAT);

        kieServicesClient = KieServicesFactory.newKieServicesClient(conf);
    }

    public ServiceResponse<ExecutionResults> executeCommands(HashMap<String, Object>
    fatos,
        String processo) {

        System.out.println("== Sending commands to the server ==");

        KieCommands commandsFactory = KieServices.Factory.get().getCommands();

```

```

ArrayList<Command<?>> commands = new ArrayList<Command<?>>();
//cria lista de inserts
fatos.forEach((id, fato) -> commands.add(commandsFactory.newInsert(fato, id)));

//verifica se existe ID
if(processo != null) {
    commands.add(commandsFactory.newStartProcess(processo));
}

//insere comando para executar regras
commands.add(commandsFactory.newFireAllRules());

Command<?> batchCommand = commandsFactory.newBatchExecution(commands, "de-
faultStatelessKieSession");

RuleServicesClient rulesClient = kieServicesClient.getServicesClient(Rule-
ServicesClient.class);

//Executa regras
ServiceResponse<ExecutionResults> executeResponse = rulesClient.execute-
CommandsResults(CONTAINER_ID, batchCommand);

return executeResponse;
}
}

```

- Acrescente um JUnit teste passando um fato conta para ser executado no kie-server.

```

@Test
public void testExecutadoRegraNoWPelaAplicacao() {
    //container utilizado no momento
    CONTAINER_ID = "banco.regras_1.0.0-SNAPSHOT";

    //lista de fatos
    HashMap<String, Object> fatos = new HashMap<>();

    //ID do processo criado no workbench
    String processo = "RF2";

    //fato criado
    Conta conta = new Conta();
    conta.setNrConta(0);
    conta.setVlBanco(2000);

    //fato inserido à lista
    fatos.put("conta1", conta);

    System.out.println("Fato: Conta Enviada");
    System.out.println(conta.toString());
    System.out.println(">>>>>");

    //Inicialização da chamada
    initialize(fatos);

    //Execução da regra
    ServiceResponse<ExecutionResults> response = executeCommands(fatos, processo);
}

```

```

//Resposta
if(response.getType() == ResponseType.SUCCESS) {
    System.out.println("Commands executed with success! Response: ");

    System.out.println("Fato: Conta recebida");
    conta = new Conta();
    //Fato retornado
    conta = (Conta) response.getResult().getValue("conta1");
    System.out.println(conta.toString());
    System.out.println("<<<<<");
}
else {
    System.out.println("Error executing rules. Message: ");
    System.out.println(response.getMsg());
}
}

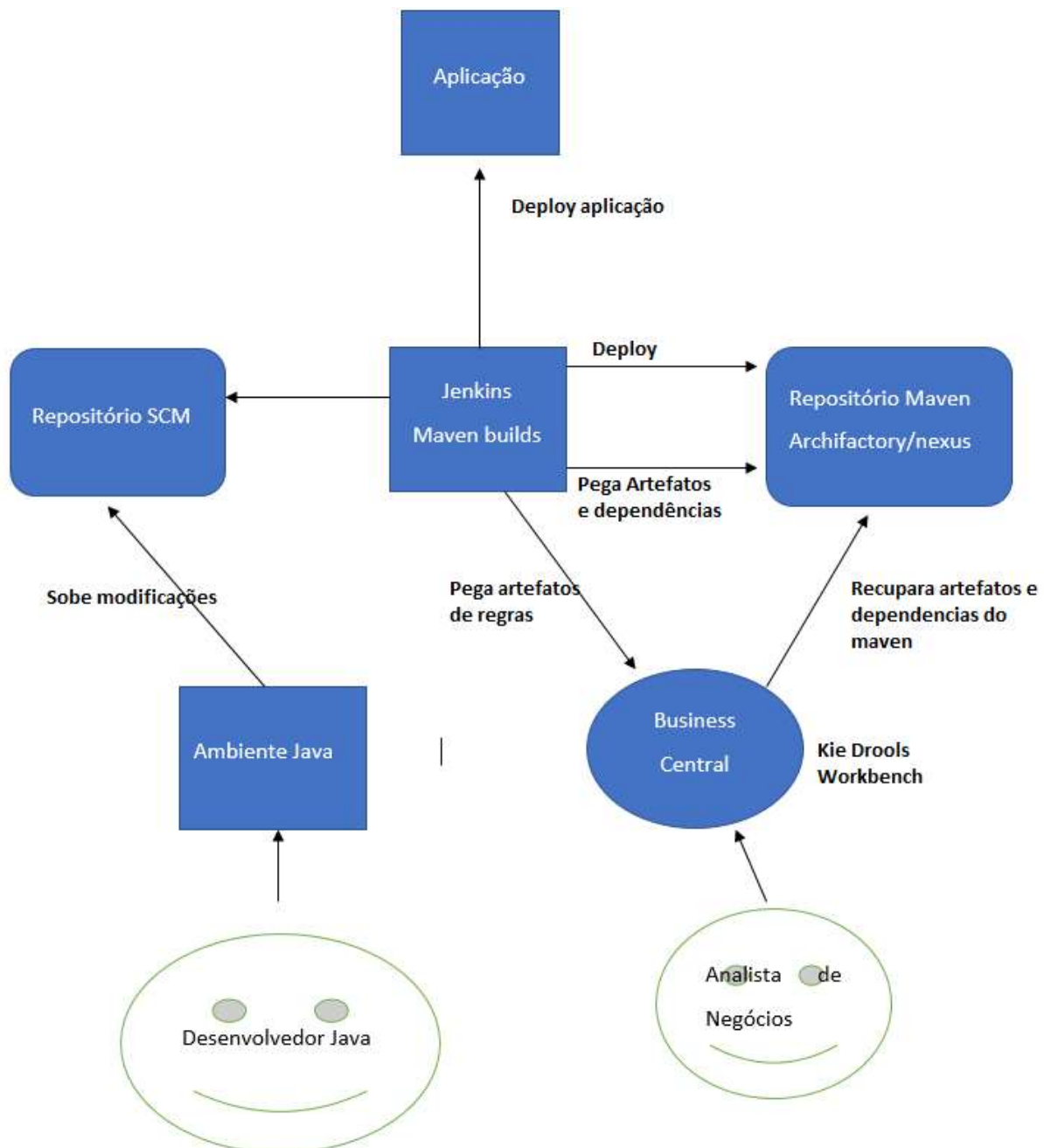
```

1.1.2. Modelo de arquitetura

Desde a versão 6.x BRMS, o maven é totalmente suportado em ambas as direções:

- O BRMS pode recuperar o artefato maven de seu repositório local, bem como de repositórios remotos.
- O BRMS pode atuar como um repositório remoto do Maven e pode ser acessado a partir de compilações externas maven.

Veja um caso típico:



- O desenvolvedor java cria a Entidade e comita o código para o repositório SCM (como um repositório git). Existe aqui uma alternativa onde isso pode ser feito no

Business Central. Neste caso, o build do maven que diz respeito ao aplicativo final irá recuperar a entidade do repositório do Business Central;

- Um “maven build” é então iniciado (no jenkins, por exemplo) e a entidade é implementada no repositório maven. O artefato maven tem um GroupId, um artefato e uma versão;
- O analista de negócios cria um novo projeto no aplicativo Business Central e, na interface com o usuário da lista de dependências, ele simplesmente insere o groupid, o artifactid e a versão que o desenvolvedor java deu a ele para entidade. O Business Central recuperará automaticamente todos os repositórios do maven remoto que foram definidos para ele o artefato da entidade, bem como todas as suas dependências;
- Ao construir o aplicativo final, o pacote de regras é recuperado por seu groupid, artifactid e versão. De fato, ao criar um projeto no Business Central, você precisa fornecer esse elemento. No arquivo de dependência do aplicativo (pom.xml), basta adicionar esses elementos de identificação, bem como a URL do repositório maven central de negócios, e ele funciona. O maven build do aplicativo recuperará a versão do pacote de regras.

Este é só um modelo de arquitetura. Outros podem ser implementados a depender da necessidade da organização.

Em teoria, não há mistério neste recurso. É apenas uma maneira de ter uma boa gestão de dependência e gerenciamento de configuração na Central de negócios. No passado, tudo isso era lidado manualmente com possíveis erros humanos. O processo agora é automatizado e está em conformidade com um padrão corporativo Apache Maven. Outra vantagem de usar o maven build é que a comunidade do Drools não precisará manter um processo de construção específico e se concentrar em outro feito.

Tudo isso deve ser configurado pela TI e não é uma preocupação do analista de negócios, exceto a versão do pacote de regras a ser usada em ambientes de desenvolvimento, integração ou produção.

Referências

- Drools, J. (s.d.). *Drools Documentation*. Acesso em 10 de 04 de 2018, disponível em Drools Documentation: https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html
- Héron, N. (s.d.). *nherondroolsonboarding*. Acesso em 01 de 04 de 2018, disponível em gitbook: <https://nheron.gitbooks.io/droolsonboarding/content/>