

CENTRO UNIVERSITÁRIO FACENS

CURSOS TECNOLÓGICOS – AS019TSN1/N2/N3



QUALIDADE E TESTES DE SOFTWARE

TÓPICOS DA AULA

BEHAVIOR-DRIVEN DEVELOPMENT

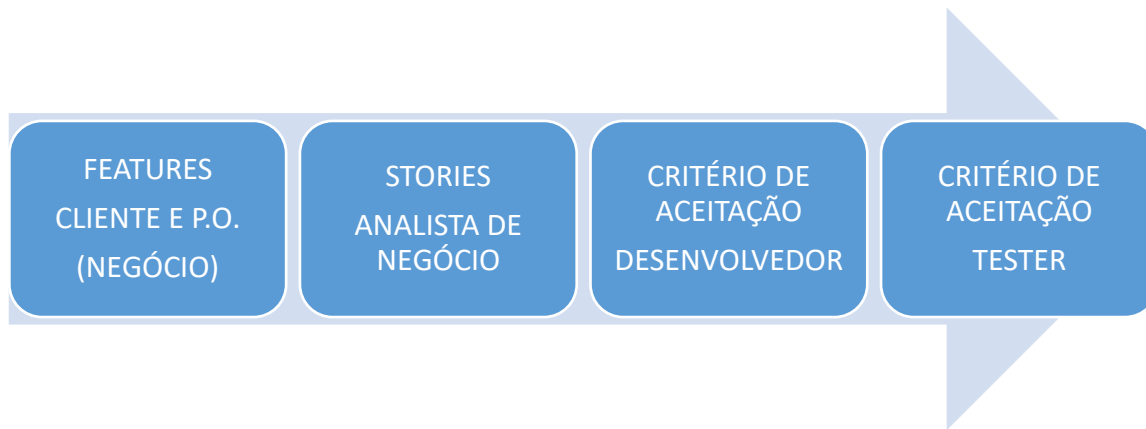
- CENÁRIO LINEAR
- CONCEITO
- CENÁRIO BDD

BDD NA PRÁTICA

- GHERKIN
- STORY
- AMBIENTE DE TESTE
- CRIANDO ARQUIVO DE TESTE
- REPRODUZIR O CENÁRIO
- CRIAÇÃO DA CLASSE DO PROJETO
- TESTANDO A IMPLEMENTAÇÃO
- DOCUMENTAÇÃO DA CLASSE

EXERCÍCIO

BEHAVIOR DRIVEN DEVELOPMENT



1. **Elicitação de Requisitos;**
2. **Desenvolvimento de Software;**
3. **Testes Manuais e Automatizados;**
4. **Revisões e Validações;**
5. **Integração e**
6. **Testes de Sistema;**
7. **Entrega;**
8. **Implantação e**
9. **Manutenção.**

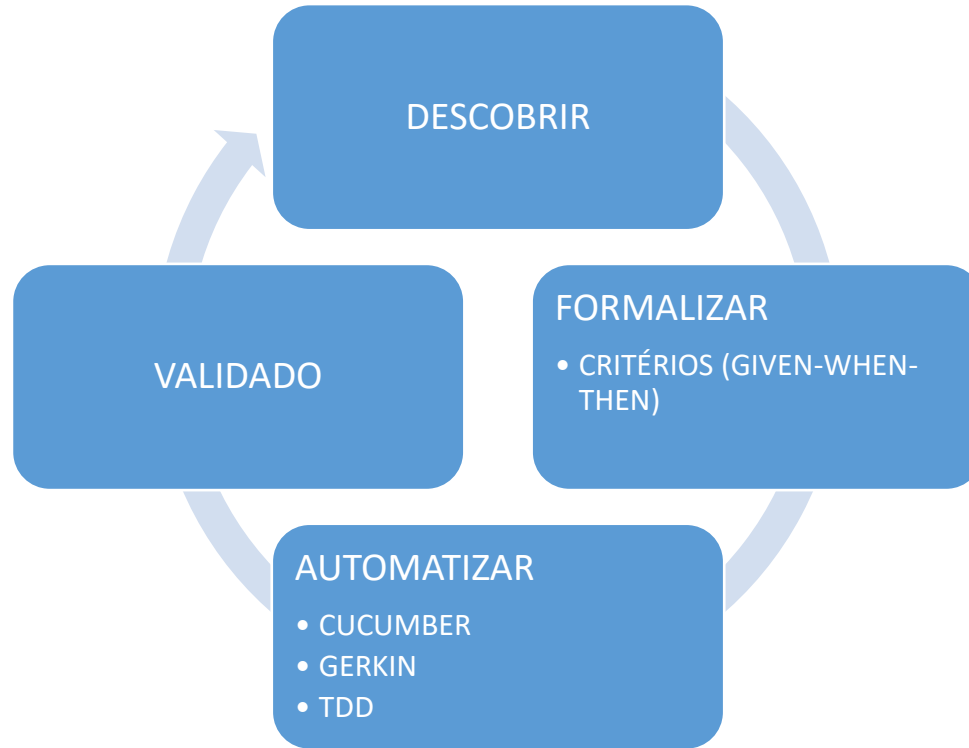
BEHAVIOR DRIVEN DEVELOPMENT

O BDD (Behavior-Driven Development) é uma abordagem de desenvolvimento de software que se concentra na descrição do comportamento de um sistema a partir da perspectiva do usuário ou do cliente. Em vez de se concentrar apenas nas funcionalidades técnicas de um software, o BDD se concentra em como o software deve se comportar em situações do mundo real.

Principais características do BDD:

- Colaboração entre Equipes;
- Documentação Executável;
- Testes Automatizados;
- Foco no Comportamento do Usuário;
- Histórias de Usuário;
- Abordagem Iterativa;
- Melhoria da Comunicação.

BEHAVIOR DRIVEN DEVELOPMENT



BEHAVIOR DRIVEN DEVELOPMENT

A aplicação do BDD (Behavior-Driven Development) envolve uma abordagem colaborativa que se concentra na definição do comportamento do software a partir da perspectiva do usuário. Para sua execução podemos seguir o exemplo abaixo:

1) Identificação de Requisitos e Histórias de Usuário:

Comece identificando os requisitos do sistema, que podem ser expressos como histórias de usuário. As histórias de usuário descrevem funcionalidades do software em termos de quem, o quê e por quê.

2) Colaboração entre Equipes:

Reúna analistas de negócios, desenvolvedores, testadores e partes interessadas para colaborar na definição do comportamento esperado do software. Promova uma compreensão comum dos requisitos e do comportamento desejado.

BEHAVIOR DRIVEN DEVELOPMENT

3) Especificação de Comportamento:

Escreva especificações de comportamento em linguagem natural ou em uma linguagem BDD, como Gherkin; As especificações devem descrever cenários de uso do software, incluindo entradas, ações do usuário e resultados esperados.

4) Testes Automatizados:

Converta as especificações de comportamento em testes automatizados. Esses testes verificam se o software se comporta de acordo com as especificações. Utilize ferramentas BDD, como o Cucumber, SpecFlow ou Behave, para criar os testes automatizados.

5) Execução de Testes:

Execute os testes automatizados regularmente durante o ciclo de desenvolvimento para garantir que o software atenda aos requisitos e ao comportamento esperado.

BEHAVIOR DRIVEN DEVELOPMENT

6) Feedback Contínuo:

Use os resultados dos testes automatizados para fornecer feedback aos desenvolvedores sobre o comportamento do software. Identifique falhas ou desvios do comportamento esperado.

7) Refinamento e Melhoria:

Se ocorrerem falhas ou se o comportamento esperado mudar, atualize as especificações de comportamento e os testes automatizados. Mantenha a documentação de comportamento atualizada para refletir as necessidades do usuário.

8) Integração com Metodologias Ágeis:

O BDD se integra bem com metodologias ágeis, como Scrum ou Kanban, fornecendo uma maneira estruturada de definir e testar requisitos incrementais.

BEHAVIOR DRIVEN DEVELOPMENT

9) Documentação Executável:

A documentação BDD serve como documentação executável, facilitando a compreensão dos requisitos e do comportamento do software para todas as partes interessadas.

10) Melhoria da Comunicação:

O BDD melhora a comunicação entre equipes e partes interessadas, garantindo que todos tenham uma compreensão clara do que o software deve fazer.

11) Iteração e Melhoria Contínua:

O BDD incentiva a iteração e a melhoria contínua à medida que novos requisitos são identificados e o software evolui.

BDD – GHERKIN

O Gherkin é uma linguagem de especificação usada principalmente em práticas de BDD (Behavior-Driven Development) e é projetada para ser legível tanto por seres humanos quanto por máquinas. É uma linguagem de formatação simples que descreve o comportamento do software em termos de cenários.

Requisito: Nome do requisito
Descrição do recurso.

Cenário: Nome do cenário
Descrição do cenário.

Given – dado um contexto;
When – quando acontecer um evento;
Then – então se espera que aconteça algo.

Cenário: Outro nome do cenário

BDD – GHERKIN

Ou podendo ser mais extenso, como no exemplo abaixo:

Requisito: Nome do requisito

Descrição do recurso.

Cenário: Nome do cenário

Descrição do cenário.

Given – dado um contexto;

When – quando acontecer um evento;

Then – então se espera que aconteça algo;

And – algo mais aconteça;

But – mas não é isso.

Cenário: Outro nome do cenário

BDD – GHERKIN

EXEMPLO DE APLICAÇÃO DO GHERKIN

Requisito: Ganhar na loteria

Ficar rico e não ter mais que me preocupar com dinheiro!

Cenário: Receber um valor mensal

Ao receber um valor mensal irá me sustentar para o resto da vida.

Como um pessoa **(beneficiado?)**

Eu quero ter uma renda mensal **(o que precisa ser feito para concluir?)**

Para que eu possa viver sem preocupar com dinheiro **(O que ganho com isso?)**

Cenário: Outro nome do cenário

BDD - EXEMPLO CONTA - STORY

Título: Cliente faz saque de dinheiro

Como um cliente, eu gostaria de sacar dinheiro em um caixa eletrônico, para que eu não tenha que esperar em uma fila do banco.

Como um cliente especial, poderei fazer saques, mesmo que o meu saldo da conta esteja negativa.

Como um cliente comum, não poderei fazer saques se o saldo da conta estiver negativo.

BDD - STORY

Story: Cliente faz saque de dinheiro Como um cliente, eu gostaria de sacar dinheiro em caixa eletrônico, para que eu não tenha que esperar numa fila de banco.

Scenario: Cliente especial com saldo negativo

Given um cliente especial com saldo atual de -200 reais

When for solicitado um saque no valor de 100 reais

Then deve efetuar o saque e atualizar o saldo da conta para -300 reais

Scenario: Cliente comum com saldo negativo

Given um cliente comum com saldo atual de -300 reais

When solicitar um saque de 200 reais

Then não deve efetuar o saque e deve retornar a mensagem Saldo Insuficiente

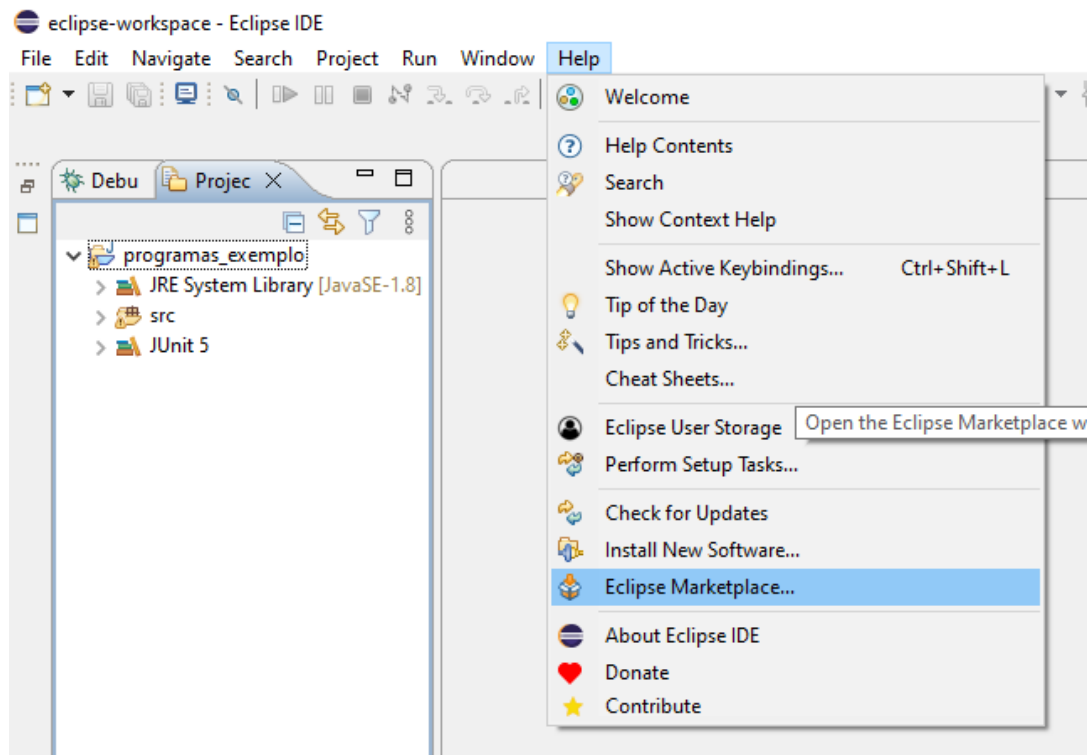
AMBIENTE DE TESTE

- **JAVA 8 (INSTALAÇÃO DO JAVA)**
- www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html
- **ECLIPSE IDE for JAVA Developers**
- <https://www.eclipse.org/downloads/packages/release/2020-03/r/eclipse-ide-java-developers>
- Faça as instalações e abra o ECLIPSE IDE;

AMBIENTE DE TESTE

CUCUMBER ECLIPSE PLUGIN

1) No Eclipse IDE selecione na **barra de Menu a opção** Help > Eclipse Marketplace;



AMBIENTE DE TESTE

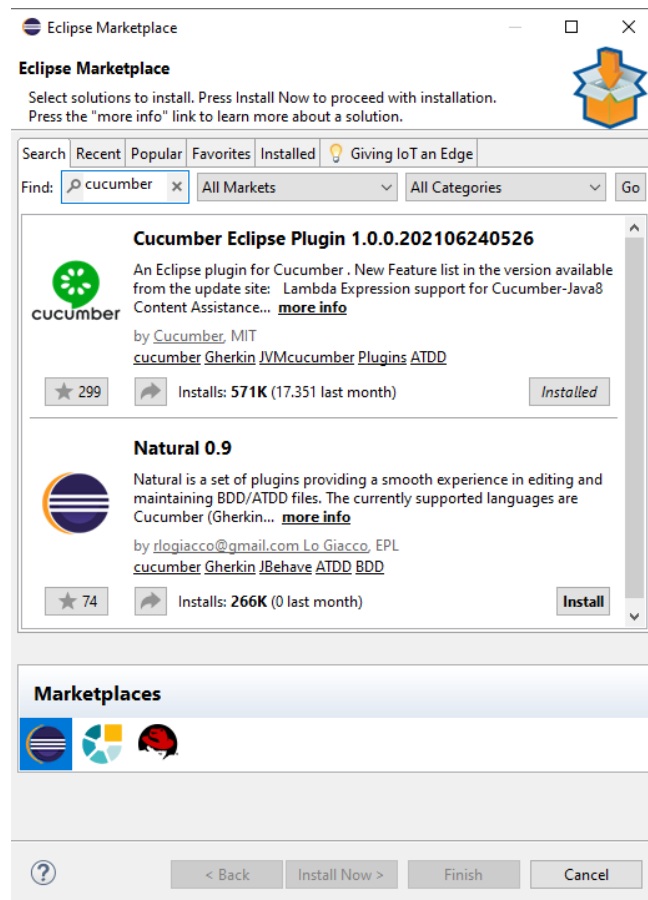
CUCUMBER ECLIPSE PLUGIN

2) No campo de Pesquisa (Find) digite “**cucumber**”;

3) Escolha o Plugin **Cucumber Eclipse Plugin** “*versão disponível*”;

*Neste momento caso apareça alguma informação de instalação

“**Aceite**” e siga com o processo até solicitar o “**Reinício da IDE**”;



AMBIENTE DE TESTE

O Cucumber é uma ferramenta de automação de testes e uma biblioteca que é amplamente utilizada em práticas de BDD (Behavior-Driven Development). Ele permite que equipes de desenvolvimento e testes criem testes automatizados com base em especificações escritas em linguagem natural, como Gherkin, e verifica se o software se comporta de acordo com essas especificações.

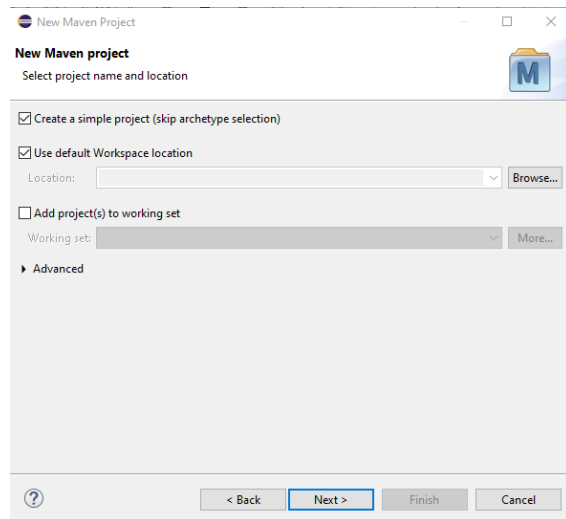
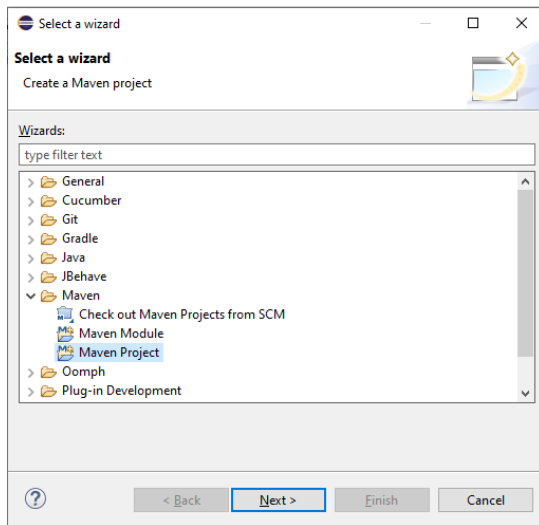
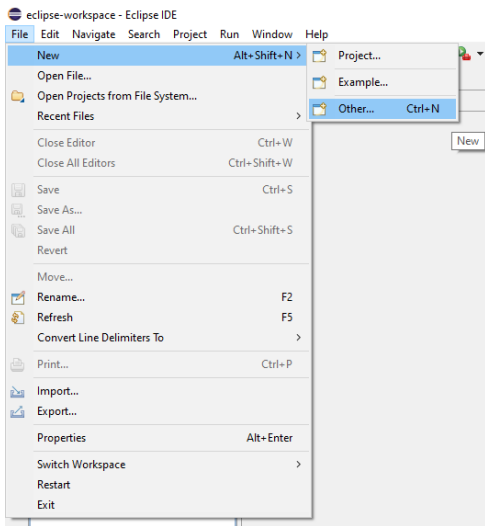


O Cucumber utiliza "tags" que são rótulos que você pode associar a cenários ou características (features) em suas especificações Gherkin. As tags ajudam a organizar e marcar os cenários, permitindo que você execute apenas os cenários marcados com tags específicas. Isso é útil quando você deseja segmentar a execução de testes com base em critérios específicos, como cenários de regressão, testes de aceitação, entre outros.

AMBIENTE DE TESTE

CRIANDO O PROJETO

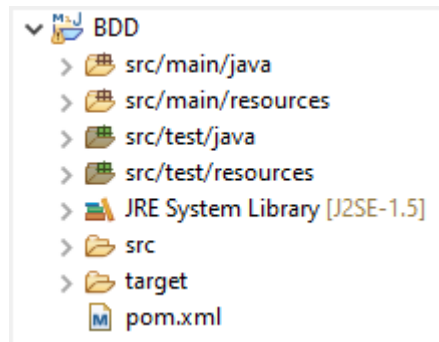
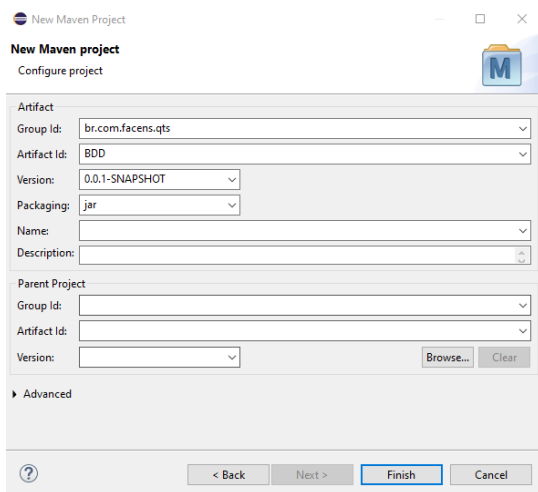
- 1) Na barra de **Menu** selecione File > New > Other;
- 2) Selecione a opção **Maven Project**;
- 3) Na configuração do novo projeto, marque a opção: **Create a simple project** e siga com **NEXT**.



AMBIENTE DE TESTE

CRIANDO O PROJETO

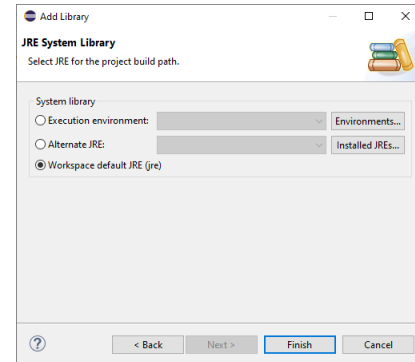
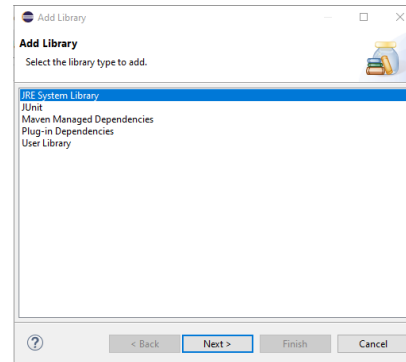
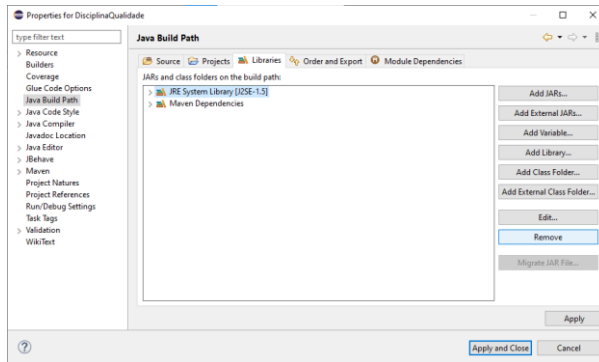
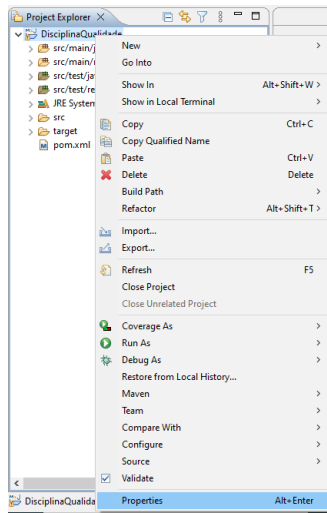
- 1) Precisamos colocar o projeto em grupo id e artefato id, para isso insira o seguintes dados:
- 2) No campo **Group Id:** br.com.facens.qts e no **Artifact Id:** BDD, após isso clique em **FINISH**;
- 3) Na aba **Project Explore** teremos o projeto construído;



AMBIENTE DE TESTE

CONFIGURAÇÃO

- 1) Selecione a pasta do projeto e clique com botão da direita do mouse, nas opções selecione a **PROPERTIES**. Nesta janela, selecione **JAVA BUIL PATH**
- 2) Selecione a opção **JRE System Library [J2SE-1.5]** e Remova o item;
- 3) Adicione uma nova **Library...** (botão **Add Library**) e selecione **JRE System Library**;
- 4) Selecione (botão) **Finish** e (botão) **Apply and Close**.



AMBIENTE DE TESTE

CONFIGURAÇÃO DO MAVEN

- 1) Precisamos configurar o maven do projeto para isso teremos que mexer no código do arquivo pom.xml;



AMBIENTE DE TESTE

CONFIGURAÇÃO DO MAVEN

- 1) Utilize o navegador e acesse o website <https://mvnrepository.com>;
- 2) Busque por **Cucumber** e selecione o arquivo **Cucumber JVM: Java**, escolha a distribuição: info.cukes. Na opção escolhida copie o código, iremos utilizar dentro do arquivo pom.xml do projeto.

The screenshot shows the Maven Repository search results for 'cucumber jvm: Java'. The search bar contains 'cucumber jvm: Java' and the results are sorted by relevance. The first result is 'Cucumber JVM: Java' by io.cucumber, with 437 usages and a MIT license. The second result is 'Cucumber JVM: Java' by info.cukes, with 297 usages and a MIT license. The third result is 'Cucumber JVM: Java 8' by io.cucumber, with 66 usages and a MIT license. The second result is highlighted with a red box.

Repository	Count
Central	50.9k
Sonatype	11.3k
Spring Plugins	5.2k
Spring Lib M	4.8k
JCenter	2.6k
Clojars	1.1k
Spring Lib Release	1.0k
IBiblio	1.0k

Group	Count
com.github	5.1k
io.github	3.8k
com.aliyun	1.4k
org.apache	1.2k
com.google	687
io.opentelemetry	681
cn.com.antcloud	554

The screenshot shows the Cucumber JVM: Java 1.2.6 artifact page. The page displays the license (MIT), date (Nov 10, 2019), files (View All), repositories (Central), ranking (#1574 in MvnRepository), and used by (297 artifacts). A note indicates that the artifact was moved to io.cucumber. The Maven tab is selected, and the dependency code is highlighted with a red box.

Home » info.cukes » cucumber-java » 1.2.6

Cucumber JVM: Java » 1.2.6
Cucumber JVM: Java

License	MIT
Date	Nov 10, 2019
Files	View All
Repositories	Central
Ranking	#1574 in MvnRepository (See Top Artifacts)
Used By	297 artifacts

Note: This artifact was moved to:
io.cucumber »

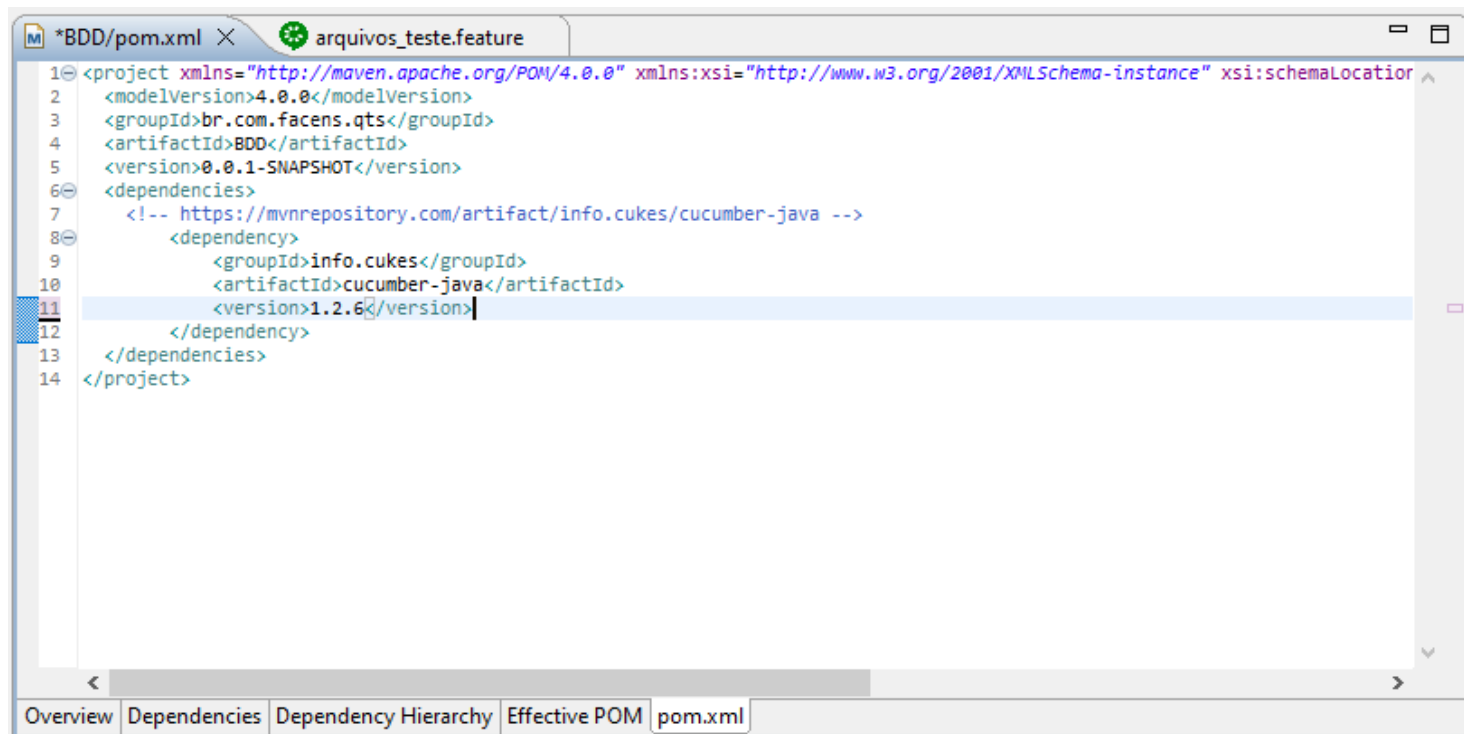
```
<!-- https://mvnrepository.com/artifact/info.cukes/cucumber-java -->
<dependency>
  <groupId>info.cukes</groupId>
  <artifactId>cucumber-java</artifactId>
  <version>1.2.6</version>
  <type>pom</type>
</dependency>
```

Include comment with link to declaration

AMBIENTE DE TESTE

CONFIGURAÇÃO DO MAVEN

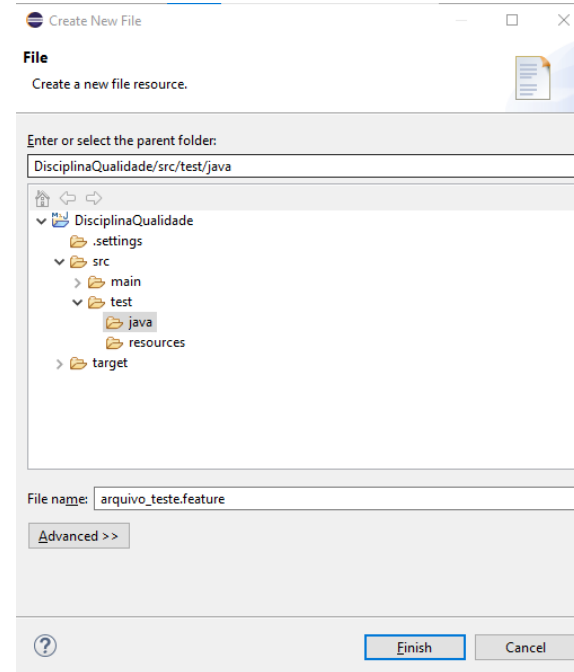
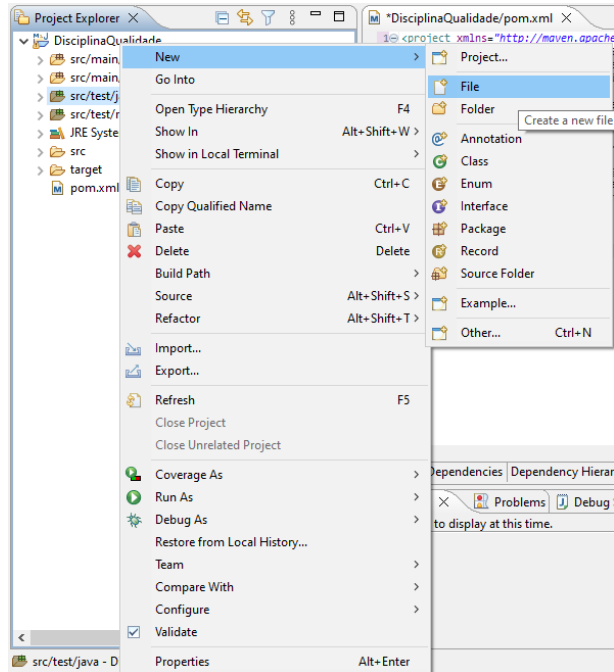
- 1) Acesse novamente o arquivo **pom.xml** do projeto;
- 2) Insira o código copiado do MVNRepository, atenção ao inserir o código.



CRIANDO ARQUIVO DE TESTE

CRIANDO UM ARQUIVO DE TESTE

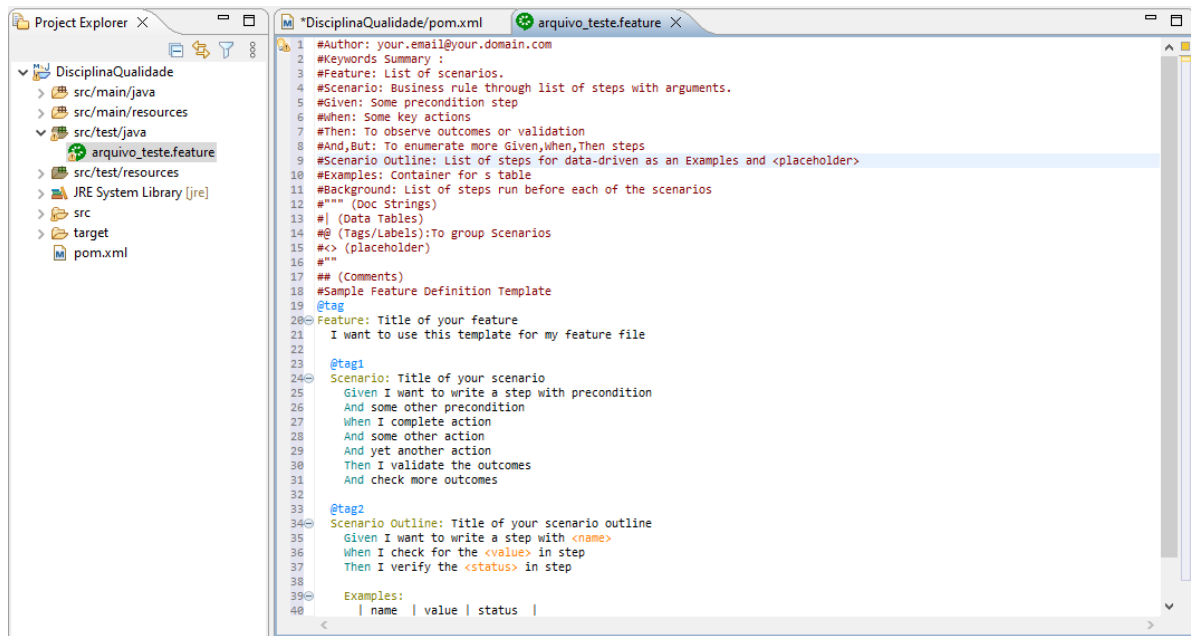
- 1) Selecione e clique com botão direito, na pasta de test/java e siga com mouse em **NEW** e depois **FILE**;
- 2) Em seguida nomeie o arquivo (ex.: arquivos_teste,feature) e clique no botão **FINISH**.



CRIANDO ARQUIVO DE TESTE

CONFIGURANDO O CONTEÚDO DO ARQUIVO DE TESTE

- 1) Após a criação do arquivo, abra o mesmo. Perceba que o ícone do arquivo criado estará diferente do tradicional.
- 2) Seu conteúdo também seguirá com um formato diferente, obedecendo as tags do cucumber.



CRIANDO ARQUIVO DE TESTE

CONFIGURANDO O CONTEÚDO DO ARQUIVO DE TESTE

- 1) Neste trecho do Código, temos a documentação e instruções do arquivo criado, utilize alterando as informações para o projeto que iremos criar **(linha 1 até linha 18)**;
- 2) No outro trecho do código será inserido os cenários **(linha 19 até linha 43)**.

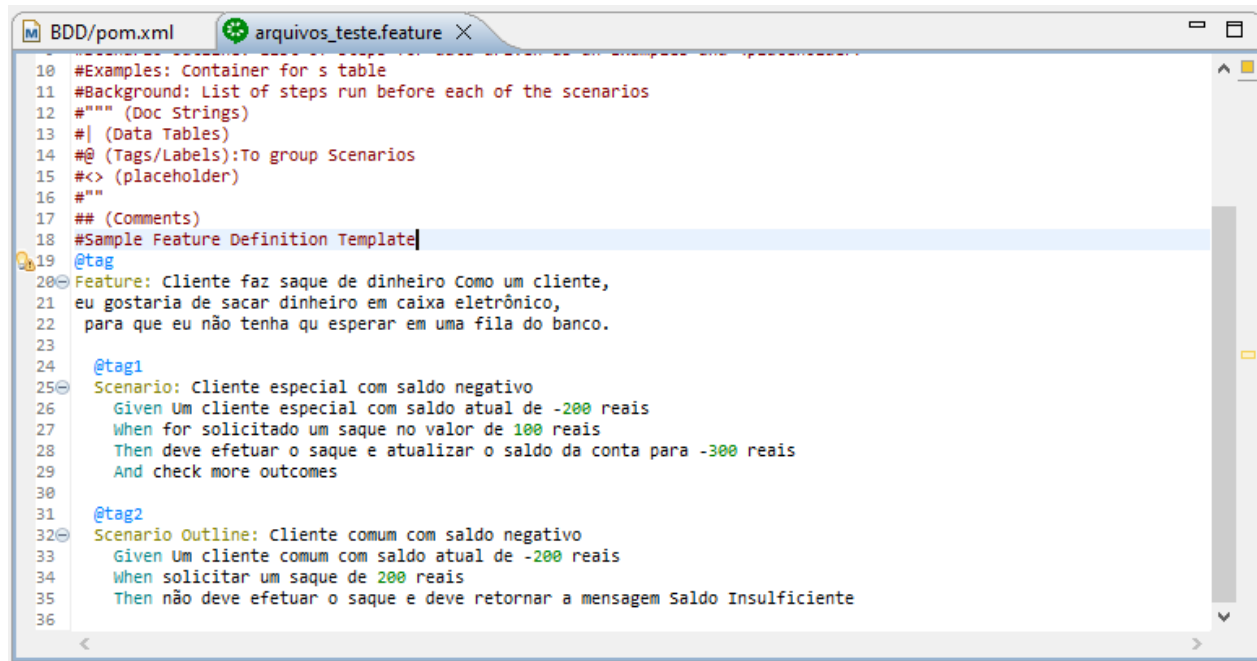
```
1 #Author: your.email@your.domain.com
2 #Keywords Summary :
3 #Feature: List of scenarios.
4 #Scenario: Business rule through list of steps with arguments.
5 #Given: Some precondition step
6 #When: Some key actions
7 #Then: To observe outcomes or validation
8 #And,But: To enumerate more Given,When,Then steps
9 #Scenario Outline: List of steps for data-driven as an Examples and <placeholder>
10 #Examples: Container for s table
11 #Background: List of steps run before each of the scenarios
12 #"" (Doc Strings)
13 #| (Data Tables)
14 #@ (Tags/Labels):To group Scenarios
15 #<> (placeholder)
16 #""
17 ## (Comments)
18 #Sample Feature Definition Template
```

```
19 @tag
20 Feature: Title of your feature
21 I want to use this template for my feature file
22
23 @tag1
24 Scenario: Title of your scenario
25 Given I want to write a step with precondition
26 And some other precondition
27 When I complete action
28 And some other action
29 And yet another action
30 Then I validate the outcomes
31 And check more outcomes
32
33 @tag2
34 Scenario Outline: Title of your scenario outline
35 Given I want to write a step with <name>
36 When I check for the <value> in step
37 Then I verify the <status> in step
38
39 Examples:
40 | name | value | status |
41 | name1 | 5 | success |
42 | name2 | 7 | Fail |
43
```

REPRODUZIR O CENÁRIOS

ALTERANDO O ARQUIVO DE TESTE

- 1) Insira os cenários indicados anteriormente, antes de compilar o projeto devemos configurar o arquivo de compilação (ver próximo slide).



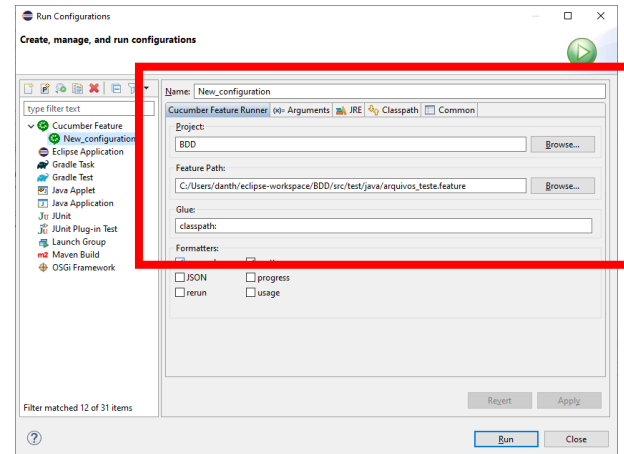
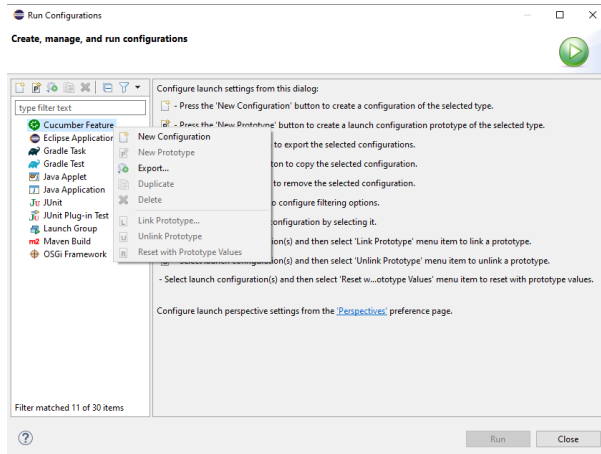
The screenshot shows a code editor window with two tabs: 'BDD/pom.xml' and 'arquivos_teste.feature'. The 'arquivos_teste.feature' tab is active, displaying a Cucumber feature file. The file contains a sample feature definition template with various tags and steps. The code is as follows:

```
10 #Examples: Container for s table
11 #Background: List of steps run before each of the scenarios
12 #"" (Doc Strings)
13 #| (Data Tables)
14 #@ (Tags/Labels):To group Scenarios
15 #<> (placeholder)
16 #""
17 ## (Comments)
18 #Sample Feature Definition Template|
19 @tag
20 Feature: Cliente faz saque de dinheiro Como um cliente,
21 eu gostaria de sacar dinheiro em caixa eletrônico,
22 para que eu não tenha qu esperar em uma fila do banco.
23
24 @tag1
25 Scenario: Cliente especial com saldo negativo
26   Given Um cliente especial com saldo atual de -200 reais
27   When for solicitado um saque no valor de 100 reais
28   Then deve efetuar o saque e atualizar o saldo da conta para -300 reais
29   And check more outcomes
30
31 @tag2
32 Scenario Outline: Cliente comum com saldo negativo
33   Given Um cliente comum com saldo atual de -200 reais
34   When solicitar um saque de 200 reais
35   Then não deve efetuar o saque e deve retornar a mensagem Saldo Insulficiente
36
```

REPRODUZIR O CENÁRIOS

ALTERANDO O ARQUIVO DE TESTE

- 1) Acesse as configurações de compilação para isso clique na seta para baixo ao lado do botão **RUN** ou acesse na barra de menu opção **RUN>RUN CONFIGURATIONS...**
- 2) Na tela abaixo, selecione a opção Cucumber Feature e com o botão da direita, crie uma nova configuração (**New Configuration**);
- 3) Nesta opção de um nome para o teste e verifique se o projeto está correto assim como seu caminho de gravação. Por fim, clique em **RUN**.



REPRODUZIR O CENARIOS

ALTERANDO O ARQUIVO DE TESTE

- 1) O resultado obtido será apresentado na aba **CONSOLE**, atenção com a seguintes informações a seguir:
- 2) Na figura abaixo, temos os cenários e os passos que forma compilados, além disso temos um relatório informando qual é o seu atual estado. No exemplo temos um Cenário e quatro passos indefinidos.

```
@tag
Feature: Cliente faz saque de dinheiro Como um cliente,
  eu gostaria de sacar dinheiro em caixa eletr?nico,
  para que eu n?o tenha qu esperar em uma fila do banco.

@tag1
Scenario: Cliente especial com saldo negativo                                     # C:/Users/danth/eclipse-workspace/BDD/src/test/java/arquivos_teste.feature:25
  Given Um cliente especial com saldo atual de -200 reais
  When for solicitado um saque no valor de 100 reais
  Then deve efetuar o saque e atualizar o saldo da conta para -300 reais
  And check more outcomes

@tag2
Scenario Outline: Cliente comum com saldo negativo                             # C:/Users/danth/eclipse-workspace/BDD/src/test/java/arquivos_teste.feature:3
  Given Um cliente comum com saldo atual de -200 reais
  When solicitar um saque de 200 reais
  Then n?o deve efetuar o saque e deve retornar a mensagem Saldo Insulficiente

1 Scenarios (1 undefined)
4 Steps (4 undefined)
0m0.000s
```

REPRODUZIR O CENARIOS

ALTERANDO O ARQUIVO DE TESTE

- 1) Nesta outra parte temos a orientação do cucumber no que deve ser implementado dentro do projeto.
- 2) Preste atenção no destaque da figura, nela temos a orientação de que a classe a ser utilizada no projeto deve seguir este modelo. O próximo passo é criar a classe e utilizar os métodos indicados.

You can implement missing steps with the snippets below:

```
@Given("^Um cliente especial com saldo atual de -(\\d+) reais$")
public void um_cliente_especial_com_saldo_atual_de_reais(int arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^for solicitado um saque no valor de (\\d+) reais$")
public void for_solicitado_um_saque_no_valor_de_reais(int arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

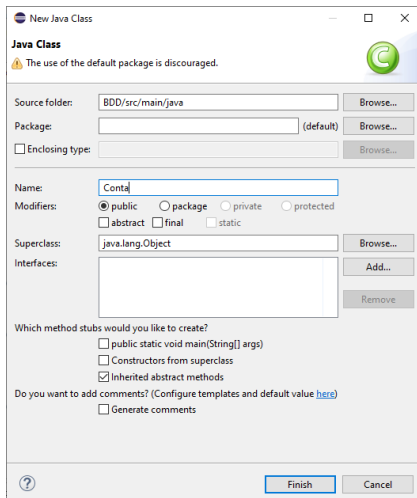
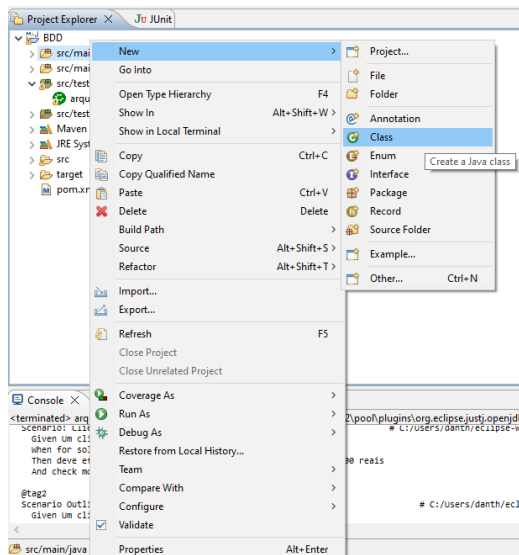
@Then("^deve efetuar o saque e atualizar o saldo da conta para -(\\d+) reais$")
public void deve_efetuar_o_saque_e_atualizar_o_saldo_da_conta_para_reais(int arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Then("^check more outcomes$")
public void check_more_outcomes() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

CRIAR CLASSE DO PROJETO

CRIANDO A CLASSE

- 1) Com os devidos pontos levantados no relatório, iremos criar uma classe para o projeto.
- 2) Após a criação insira a orientação dada pelo cucumber, será necessário realizar um import de algumas bibliotecas do cucumber para classe criada (ver próximo slide).

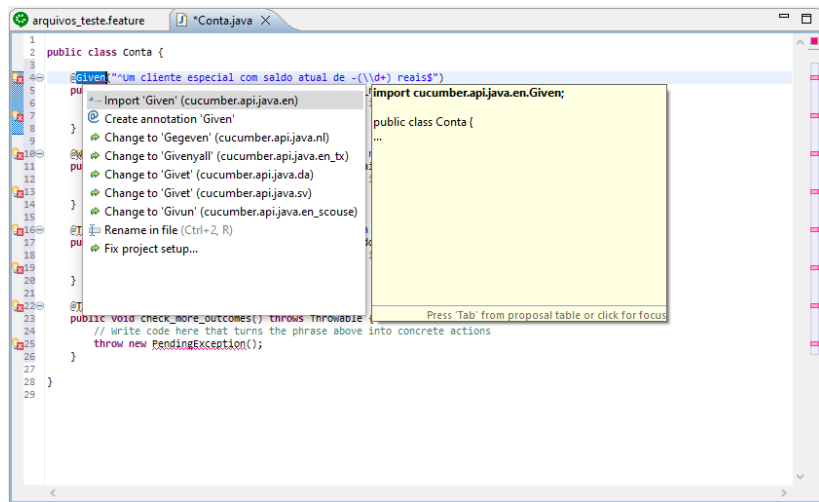


```
1 public class Conta {
2
3
4     @Given("^Um cliente especial com saldo atual de -(\\d+) reais$")
5     public void um_cliente_especial_com_saldo_atual_de_reais(int arg1) throws Throwable {
6         // Write code here that turns the phrase above into concrete actions
7         throw new PendingException();
8     }
9
10    @When("^for solicitado um saque no valor de (\\d+) reais$")
11    public void for_solicitado_um_saque_no_valor_de_reais(int arg1) throws Throwable {
12        // Write code here that turns the phrase above into concrete actions
13        throw new PendingException();
14    }
15
16    @Then("^deve efetuar o saque e atualizar o saldo da conta para -(\\d+) reais$")
17    public void deve_efetuar_o_saque_e_atualizar_o_saldo_da_conta_para_reais(int arg1) throws Throwable {
18        // Write code here that turns the phrase above into concrete actions
19        throw new PendingException();
20    }
21
22    @Then("^check more outcomes$")
23    public void check_more_outcomes() throws Throwable {
24        // Write code here that turns the phrase above into concrete actions
25        throw new PendingException();
26    }
27
28 }
29
```


CRIAR CLASSE DO PROJETO

CRIANDO A CLASSE

- 1) Utilize a IDE para identificar o erro e a proposta de solução, caso queira inserir o Código, digite as importações que estão no cabeçalho da classe. Ao término, compile o arquivo de teste e veja o resultado na aba **CONSOLE**



```
1 import cucumber.api.PendingException;
2 import cucumber.api.java.en.Given;
3 import cucumber.api.java.en.Then;
4 import cucumber.api.java.en.When;
5
6 public class Conta {
7
8     @Given("^Um cliente especial com saldo atual de -((\\d+) reais)$")
9     public void um_cliente_especial_com_saldo_atual_de_reais(int arg1) throws Throwable {
10         // Write code here that turns the phrase above into concrete actions
11         throw new PendingException();
12     }
13
14     @When("^for solicitado um saque no valor de ((\\d+) reais)$")
15     public void for_solicitado_um_saque_no_valor_de_reais(int arg1) throws Throwable {
16         // Write code here that turns the phrase above into concrete actions
17         throw new PendingException();
18     }
19
20     @Then("^deve efetuar o saque e atualizar o saldo da conta para -((\\d+) reais)$")
21     public void deve_efetuar_o_saque_e_atualizar_o_saldo_da_conta_para_reais(int arg1) throws Throwable {
22         // Write code here that turns the phrase above into concrete actions
23         throw new PendingException();
24     }
25
26     @Then("^check more outcomes$")
27     public void check_more_outcomes() throws Throwable {
28         // Write code here that turns the phrase above into concrete actions
29         throw new PendingException();
30     }
31
32 }
33
```

TESTANDO A IMPLEMENTAÇÃO

TESTANDO A CLASSE CRIADA

- 1) Os dados entregues no **CONSOLE** indicam informações diferentes do que foram mostrado anteriormente, isso se deve porque os métodos criados não executam nenhuma atividade;

```
at cucumber.runtime.Runtime.runStep(Runtime.java:300)
at cucumber.runtime.model.StepContainer.runStep(StepContainer.java:44)
at cucumber.runtime.model.StepContainer.runSteps(StepContainer.java:39)
at cucumber.runtime.model.CucumberScenario.run(CucumberScenario.java:44)
at cucumber.runtime.model.CucumberFeature.run(CucumberFeature.java:165)
at cucumber.runtime.Runtime.run(Runtime.java:122)
at cucumber.api.cli.Main.run(Main.java:36)
at cucumber.api.cli.Main.main(Main.java:18)

When for solicitado um saque no valor de 100 reais # Conta.for_solicitado_um_saque_no_valor_de_reais(int)
Then deve efetuar o saque e atualizar o saldo da conta para -300 reais # Conta.deve_efetuar_o_saque_e_atualizar_o_saldo_da_conta_pai
And check more outcomes # Conta.check_more_outcomes()

@tag2
Scenario Outline: Cliente comum com saldo negativo # C:/Users/danth/eclipse-workspace/BDD/src/test/java/a
  Given Um cliente comum com saldo atual de -200 reais
  When solicitar um saque de 200 reais
  Then n?o deve efetuar o saque e deve retornar a mensagem Saldo Insuficiente

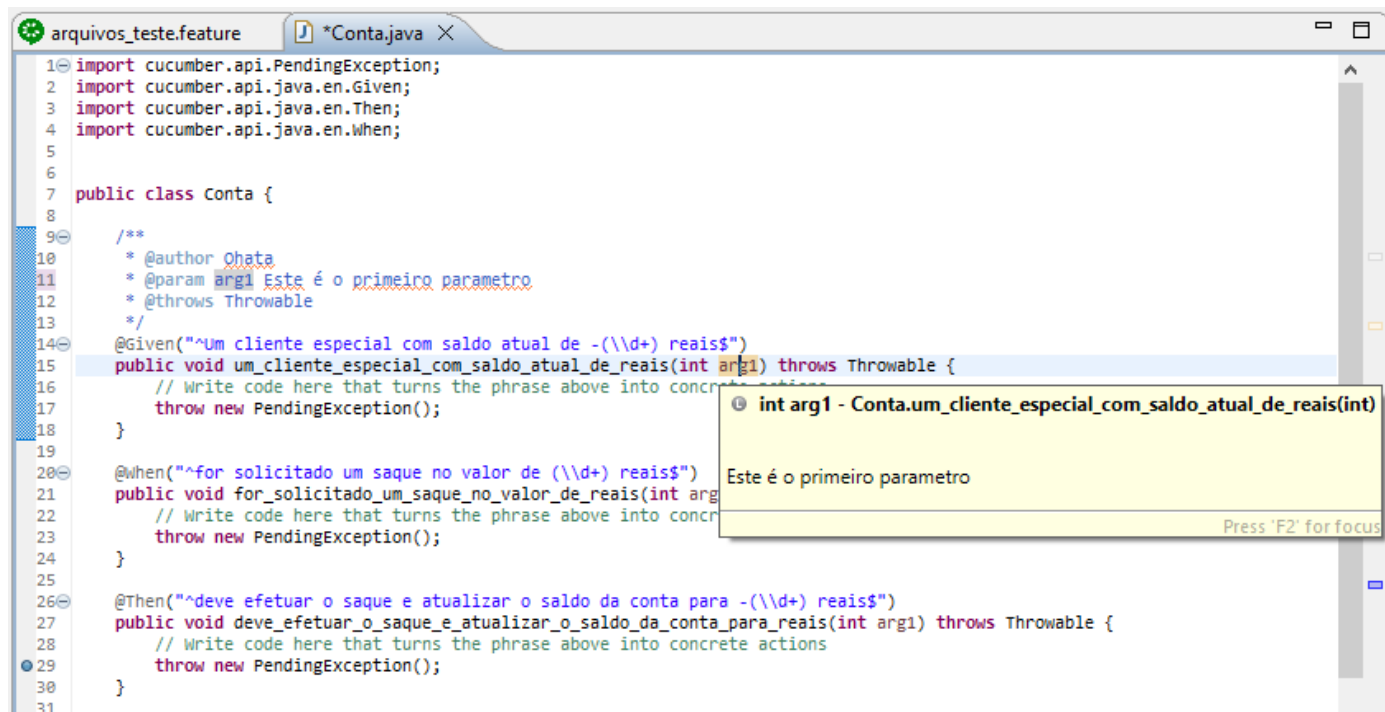
Failed scenarios:
C:/Users/danth/eclipse-workspace/BDD/src/test/java/arquivos_teste.feature:25 # Scenario: Cliente especial com saldo negativo

1 Scenarios (1 failed)
4 Steps (1 failed, 3 skipped)
0m0,366s
```

DOCUMENTAÇÃO DA CLASSE

DOCUMENTAÇÃO

- 1) Quando criamos uma classe com a indicação do cucumber, durante o processo de documentação da classe a IDE irá indicar informações que devem ser utilizadas para conter em seu documento.



UTILIZANDO CLASSE RUNNER

Em alguns casos, o projeto não poderá ser rodado diretamente pelo arquivo com a extensão **.feature**, desta forma se faz necessário criar uma classe Runner para o projeto. Para isso devemos buscar no **MVN REPOSITORY** uma dependência a mais no projeto.


The screenshot shows the Maven Repository search results for the keyword 'cucumber'. The search bar at the top contains 'cucumber' and a 'Search' button. The left sidebar shows the 'Repository' list with 'Central' selected, and the 'Group' list with 'io.cucumber' selected. The main content area displays 'Found 541 results' and a list of search results. The first result, '1. Cucumber JVM: JUnit', is highlighted with a red box. It shows the package 'info.cukes » cucumber-junit', 371 usages, and the MIT license. The second result is '2. Cucumber JVM: Java' with 368 usages and MIT license. The third result is '3. Cucumber JVM: JUnit' with 295 usages and MIT license. The results are sorted by 'relevance'.

Repository	Group	Artifact	Version	Usages	License
Central	io.cucumber	cucumber-junit	4.52	371	MIT
Central	io.cucumber	cucumber-java	1.34	368	MIT
Central	io.cucumber	cucumber-junit	5.5	295	MIT

UTILIZANDO CLASSE RUNNER

Iremos utilizar a dependência Cucumber JVM: JUnit, com ela poderemos utilizar um classe para rodar e testar o projeto. Na figura abaixo temos as opções disponíveis da dependência.

[Home](#) » [info.cukes](#) » cucumber-junit

**Cucumber JVM: JUnit**
Cucumber JVM: JUnit

License	MIT
Tags	junit testing
Ranking	#1134 in MvnRepository (See Top Artifacts)
Used By	371 artifacts

Note: This artifact was moved to:
[io.cucumber](#) »

Central (44)

	Version	Vulnerabilities	Repository	Usages	Date
1.2.x	1.2.6		Central	4	Nov 10, 2019
	1.2.5		Central	166	Sep 12, 2016
	1.2.4		Central	139	Jul 23, 2015
	1.2.3		Central	27	Jul 07, 2015

UTILIZANDO CLASSE RUNNER

Iremos utilizar a dependência Cucumber JVM: JUnit, com ela poderemos utilizar um classe para rodar e testar o projeto. Na figura abaixo temos as opções disponíveis da dependência. No exemplo será utilizado a versão 1.2.5. Isso se deve porque temos uma quantidade maior de usuário aplicando, versões mais recentes tendem a gerar alguns erros devido a falta de aplicação.

Home > info.cukes > cucumber-junit

Cucumber JVM: JUnit
Cucumber JVM: JUnit

License: MIT

Tags: junit, testing

Ranking: #1134 in MvnRepository (See Top Artifacts)

Used By: 371 artifacts

Note: This artifact was moved to:
io.cucumber >

Central (44)

Version	Vulnerabilities	Repository	Usages	Date
1.2.6		Central	4	Nov 10, 2019
1.2.5		Central	166	Sep 12, 2016
1.2.4		Central	139	Jul 23, 2015
1.2.3		Central	27	Jul 07, 2015

Cucumber JVM: JUnit » 1.2.5
Cucumber JVM: JUnit

License: MIT

Tags: junit, testing

Date: Sep 12, 2016

Files: pom (2 KB) | jar (21 KB) | View All

Repositories: Central, OneBusAway Pub, Sonatype

Ranking: #1134 in MvnRepository (See Top Artifacts)

Used By: 371 artifacts

Vulnerabilities: Vulnerabilities from dependencies: CVE-2020-15250

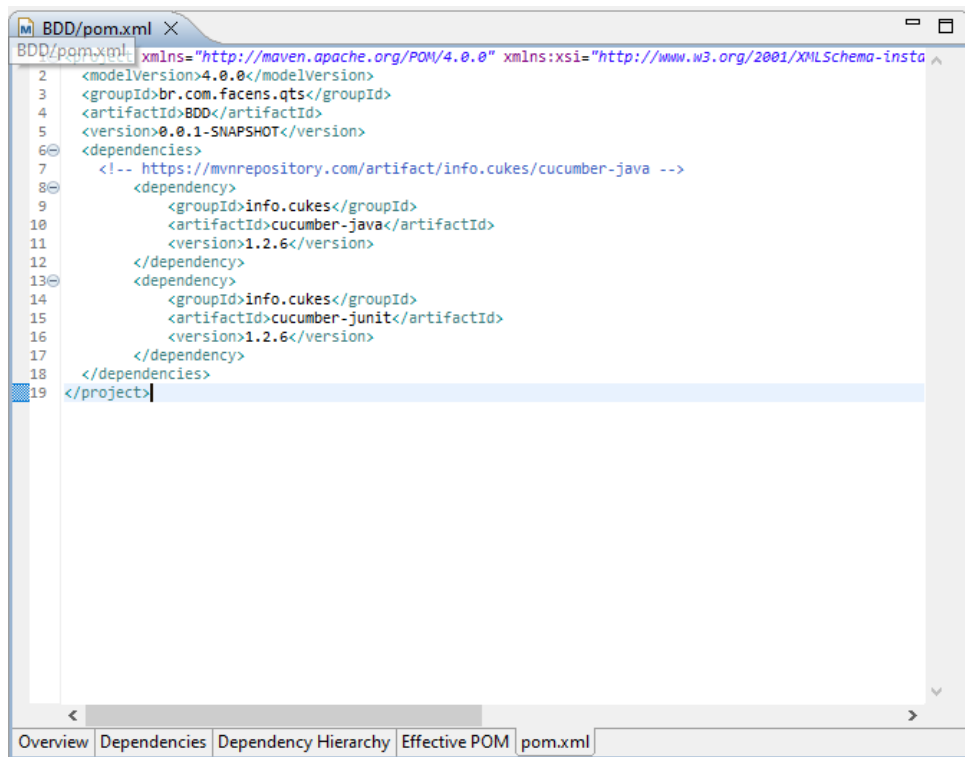
Note: There is a new version for this artifact
New Version: 1.2.6

Maven, Gradle, Gradle (Short), Gradle (Kotlin), SBT, Ivy, Grape, Leiningen, Buildr

```
<!-- https://mvnrepository.com/artifact/info.cukes/cucumber-junit -->
<dependency>
  <groupId>info.cukes</groupId>
  <artifactId>cucumber-junit</artifactId>
  <version>1.2.5</version>
  <scope>test</scope>
</dependency>
```

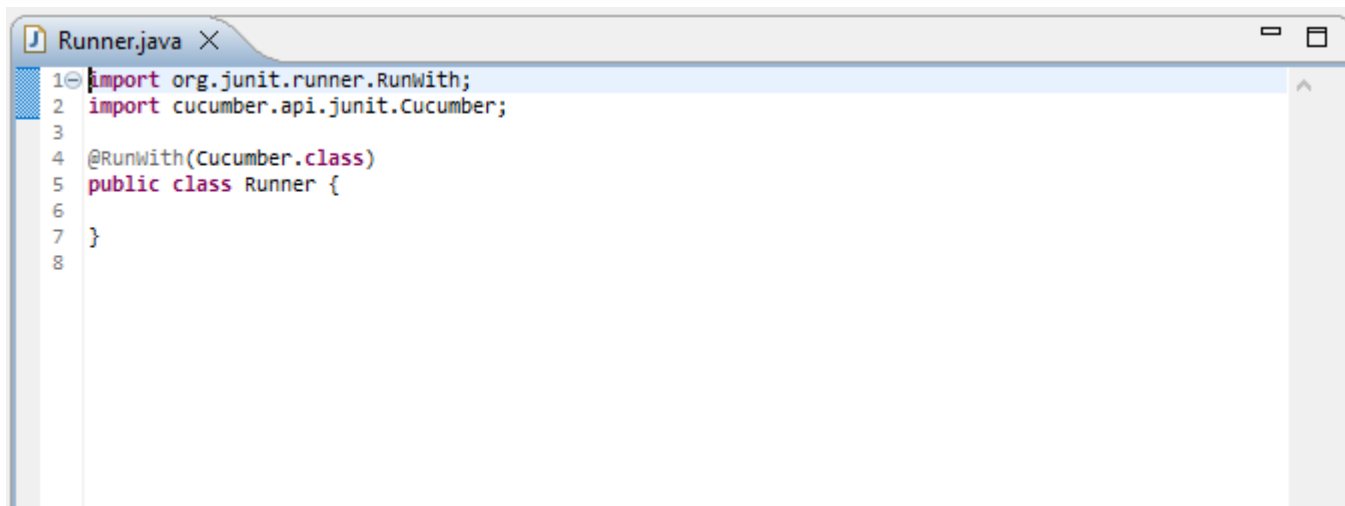
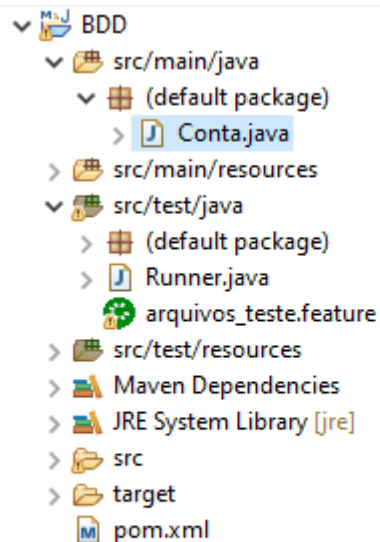
UTILIZANDO CLASSE RUNNER

Após copiar as informações da dependência devemos atualizar o arquivo pom.xml que está no projeto. Acesse o arquivo e atualize o item, conforme a figura abaixo:



UTILIZANDO CLASSE RUNNER

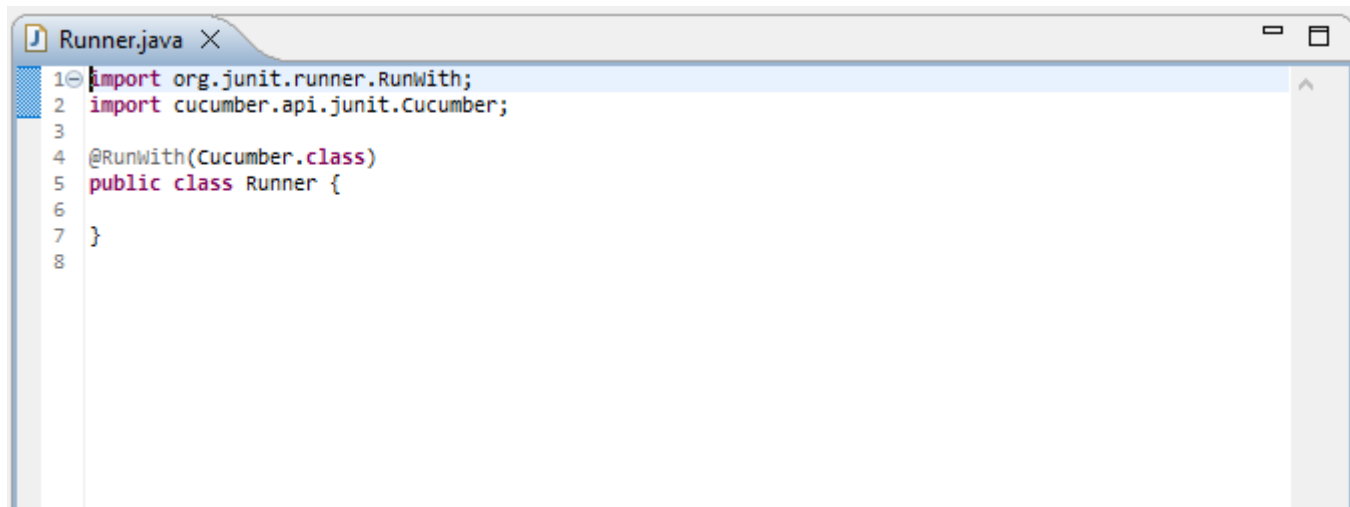
Agora precisamos de uma classe para rodar o projeto sempre que necessário. Para isso, devemos criar a classe Runner dentro da pasta de teste. Esta classe irá conter um simples código, conforme a figura abaixo:



UTILIZANDO CLASSE RUNNER

A anotação `@RunWith` é usada em testes unitários em Java com o framework JUnit, embora seu uso tenha sido substituído em versões mais recentes do JUnit por outras anotações, como `@ExtendWith`.

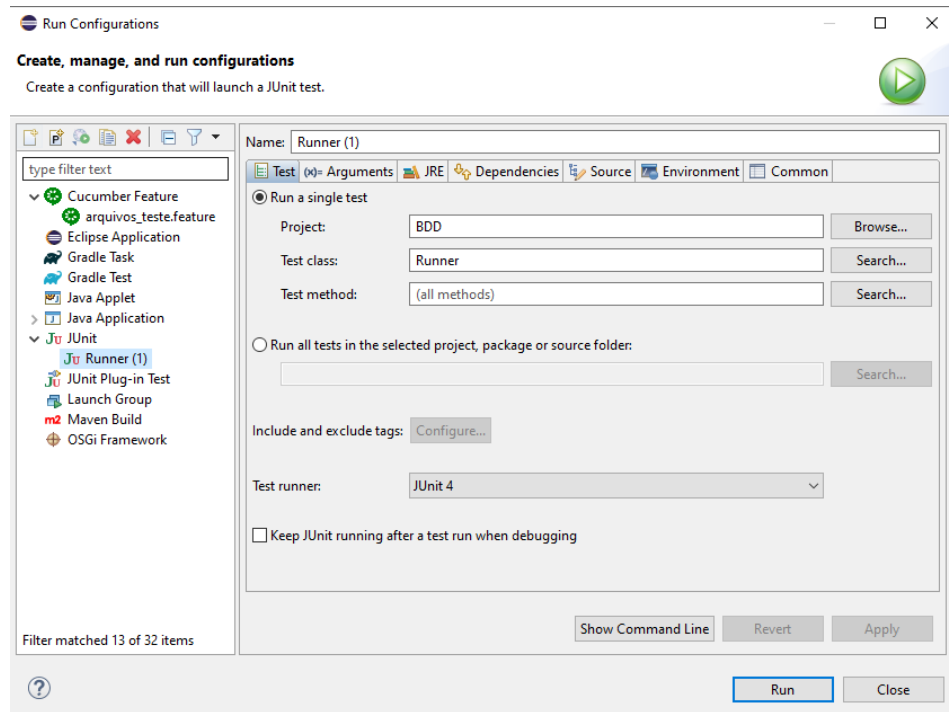
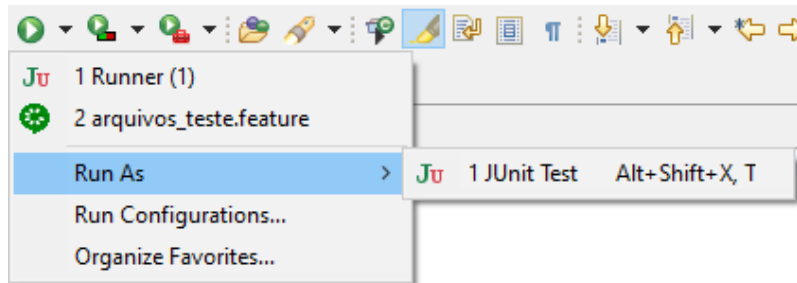
Antes das versões mais recentes do JUnit, você usaria `@RunWith` para especificar uma classe que fornece a funcionalidade de execução dos testes. Normalmente, você usaria `@RunWith` em conjunto com uma classe que implementa a interface `Runner`. O `Runner` é responsável por executar os testes em uma classe de teste específica e gerar relatórios dos resultados.



```
1 import org.junit.runner.RunWith;
2 import cucumber.api.junit.Cucumber;
3
4 @RunWith(Cucumber.class)
5 public class Runner {
6
7 }
8
```

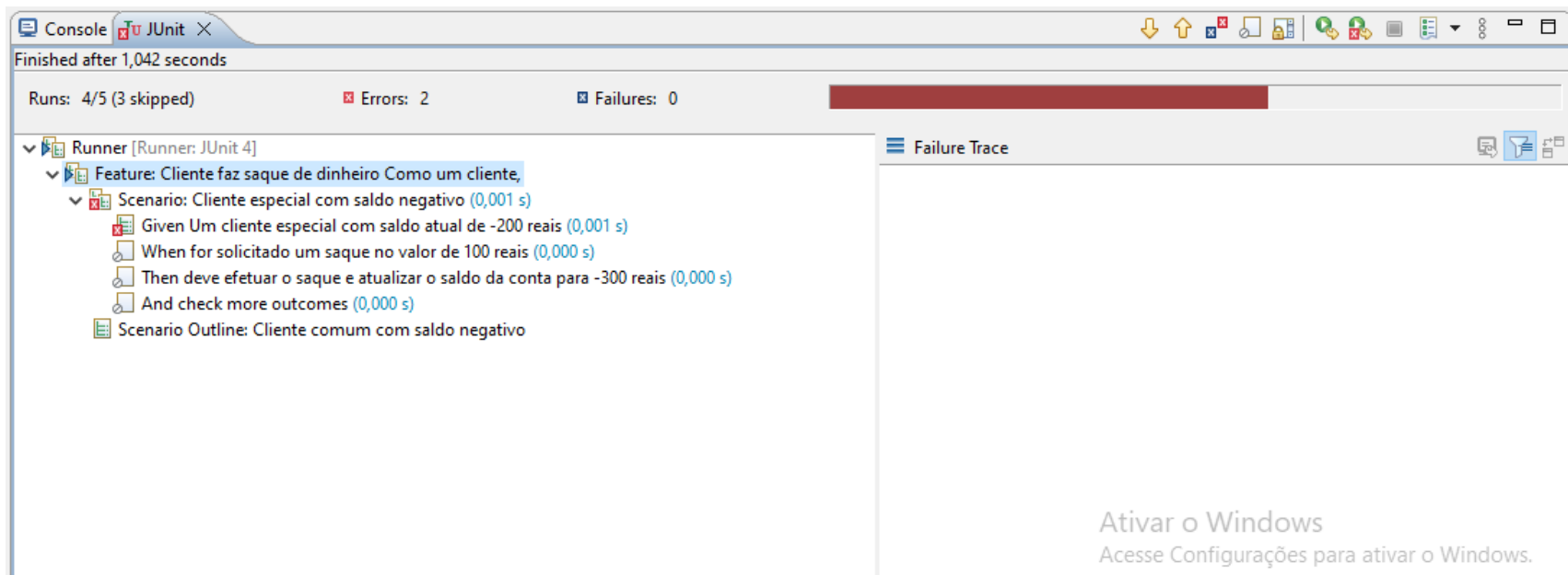
UTILIZANDO CLASSE RUNNER

Após a criação da classe Runner ao executar o projeto, veja que a mesma irá aparecer na opção de compilação.



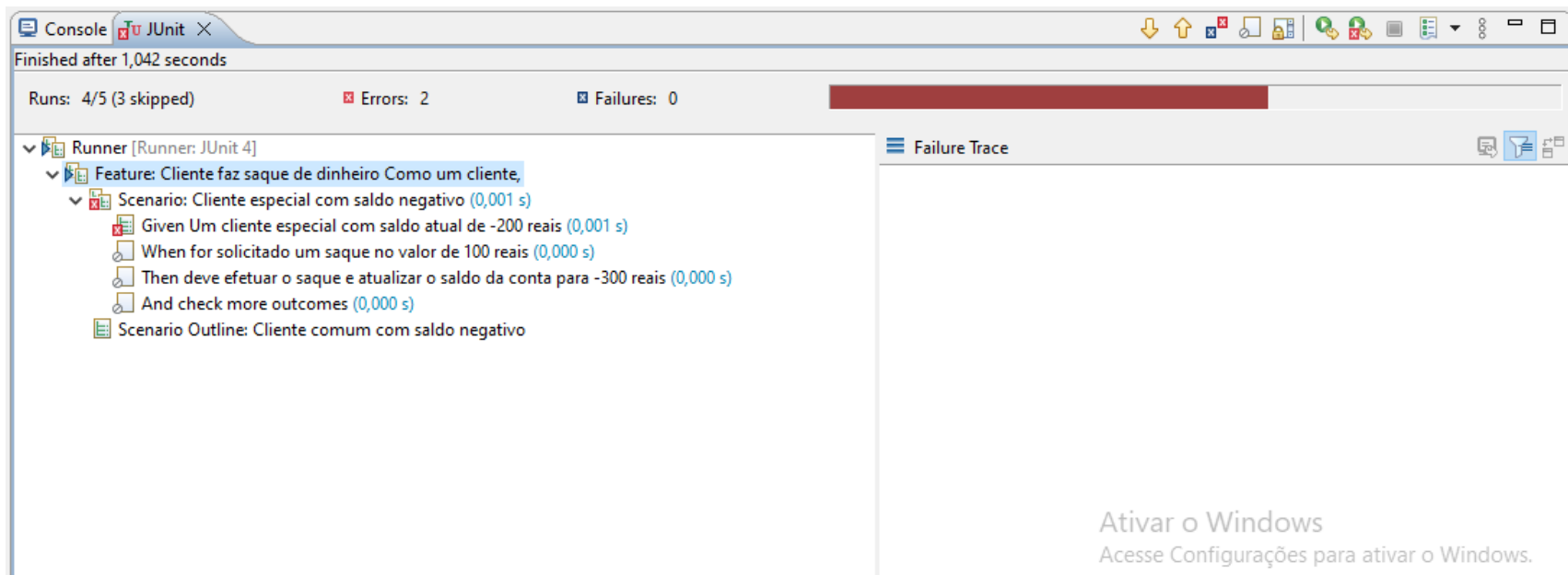
UTILIZANDO CLASSE RUNNER

Ao executar o projeto veja que a aba JUnit irá ser aberta e demonstrar os itens a serem executados para criação do projeto.



UTILIZANDO CLASSE RUNNER

Ao executar o projeto veja que a aba JUnit irá ser aberta e demonstrar os itens a serem executados para criação do projeto.



EXERCÍCIO

EXERCÍCIO INDIVIDUAL CONTA_TESTE

REPRODUZA O EXEMPLO DADO CONFORME EXPLICADO EM AULA;

CRIE UM REPOSITÓRIO REMOTO DO PROJETO, ESCREVA O README.MD CONFORME O EXEMPLO DADO NO CONTEÚDO DA AULA;

ESCREVA O CÓDIGO DA **CLASSE CONTA** UTILIZANDO OS MÉTODOS QUE O CUCUMBER INDICOU;

AO TÉRMINO, EXECUTE O PROJETO E VERIFIQUE AS INFORMAÇÕES DADAS NO CONSOLE, TIRE PRINT DESTAS INFORMAÇÕES E UTILIZE NA CONSTRUÇÃO DO README.MD DO REPOSITÓRIO REMOTO DO PROJETO;

APÓS TER TODAS AS ETAPAS CRIADAS, CRIE A DOCUMENTAÇÃO DA CLASSE CONTA E DO ARQUIVO DE TESTE CUCUMBER (USE O EXEMPLO QUE A BIBLIOTECA GERA NO ARQUIVO), GERE UM JAVA DOC DO PROJETO;

O EXERCÍCIO FAZ PARTE DAS ATIVIDADES FINAIS E DEVE SER ENTREGUE NO 28/11 ATÉ ÀS 23:59, ESTE PRAZO NÃO SERÁ PRORROGADO;

O EXERCÍCIO É INDIVIDUAL E EM CASO DE CÓPIA AMBOS OS ALUNOS RECEBERAM ZERO;

ENVIE O LINK DO SEU REPOSITÓRIO REMOTO NO EXERCÍCIO;

UTILIZE À AULA PARA TIRAR DÚVIDAS E REALIZAR O EXERCÍCIO.

PERGUNTAS?



daniel.ohata@facens.br

MUITO OBRIGADO!!!!



daniel.ohata@facens.br