

TestNG Test Framework

Definição

O TestNG é um framework, em Java e *open source*, de testes automatizados[1]. É inspirado em NUnit e JUnit, porém com adição de algumas funcionalidades que o tornam mais poderoso e mais fácil de usar[2], como agrupamento, dependência, priorização, múltiplas anotações, etc. Abrange uma gama mais ampla de categorias de teste, como os de unidade, funcional, *end to end*, de integração, etc. Uma outra razão para tornar essa ferramenta tão popular entre desenvolvedores e testadores, para criação de testes, é o fato de que ele ajuda na organização dos mesmos de forma estruturada, melhorando a capacidade de manutenção e legibilidade dos scripts[1].

Ele pode ser instalado no computador de várias formas, sendo algumas delas: usando a IDE Eclipse, baixando pelo Marketplace; usando Maven ou Gradle, adicionando o TestNG como dependência do arquivo de configuração do projeto; de forma manual, diretamente do site <http://testng.org/doc/download.html>, sendo adicionado como uma biblioteca externa; e, fazendo a integração com outras ferramentas de automação de teste.

A seguir, serão listadas algumas das vantagens e desvantagens do uso dessa ferramenta.

Vantagens e Desvantagens

Como vantagens, podemos citar:

- Configuração flexível: O TestNG oferece opções de configuração extensivas, permitindo a configuração do ambiente de execução de teste de acordo com as devidas necessidades;
- Anotações: O TestNG faz uso de anotações (*annotations*), o que torna mais fácil a marcação de métodos de teste, definição de dependências de teste, definição de prioridades e agrupamento de testes. Isso significa um código de teste mais legível e fácil de manter;
- Execução Paralela: O TestNG suporta a execução paralela de testes, o que pode acelerar significativamente o processo de teste, especialmente em projetos maiores com um grande número de testes;
- Agrupamento e Priorização: Com o TestNG, é possível organizar testes em grupos e definir a ordem de execução usando prioridades. Isso ajuda a focar em áreas específicas de teste e garantir que testes críticos sejam executados primeiro;
- Testes com Dados Variados: O TestNG suporta testes baseados em dados, permitindo que você execute o mesmo teste com diferentes conjuntos de dados. Isso é útil para testar vários cenários com mínima duplicação de código;
- Ouvintes e Relatórios: O TestNG fornece ouvintes que permitem o acesso às várias etapas do processo de execução de teste. Além disso, ele gera relatórios detalhados em HTML e XML, tornando mais fácil a identificação de resultados e falhas nos testes;

- Integração: O TestNG integra-se perfeitamente a outras ferramentas e estruturas como Maven, Gradle, Jenkins e várias IDEs, tornando-o adequado para diferentes ambientes de desenvolvimento;
- Recursos Incorporados: O TestNG oferece recursos incorporados como tempos limite, injeção de dependência e parametrização, que ajudam na criação de testes abrangentes e eficientes.

Como desvantagens, podemos citar:

- Curva de Aprendizado: Embora os conceitos principais do TestNG sejam relativamente fáceis de entender, suas funcionalidades mais avançadas, como estratégias de execução paralela e opções de configuração complexas, podem ter uma curva de aprendizado mais íngreme para iniciantes;
- Documentação: Embora o TestNG tenha uma documentação decente, alguns casos de uso específicos podem exigir uma exploração mais profunda da documentação ou dos fóruns de comunidade para encontrar soluções adequadas;
- Tamanho da Comunidade: Embora o TestNG tenha uma comunidade sólida, ela pode não ser tão grande quanto algumas outras estruturas de teste, como o JUnit. O que pode significar que existe um número ligeiramente menor de recursos online e suporte, se compararmos com estruturas que são mais amplamente usadas;
- Código Legado: Se você estiver trabalhando com código legado, que já está construído no JUnit, migrar para o TestNG pode exigir algum esforço e alterações em sua suíte de testes existente;
- Desafios de Integração: Embora o TestNG seja projetado para integrar bem com outras ferramentas, algumas integrações ainda podem exigir configuração adicional e solução de problemas.

Tutorial

No seguinte tutorial, será mostrada a instalação do TestNG na IDE Eclipse, via Marketplace e, logo em seguida, será mostrada uma aplicação da ferramenta em um código simples de um sistema de cadastro de livros em uma biblioteca.

1º Passo: Para a instalação do TestNG, deve-se ter a IDE Eclipse e o Java instalado. A interface da IDE Eclipse é mostrada na Figura 1.

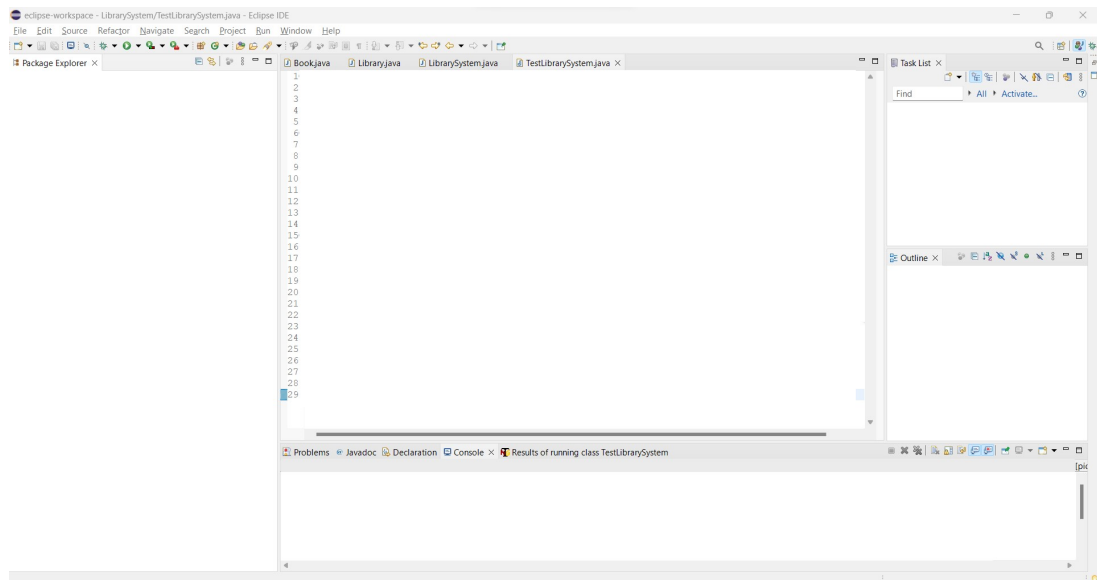


Figura 1 - Interface da IDE Eclipse

2º Passo: Para acessar o Marketplace da IDE, deve-se clicar em 'Help' na barra de ferramentas e selecionar a opção 'Eclipse Marketplace...', conforme indicado na Figura 2.

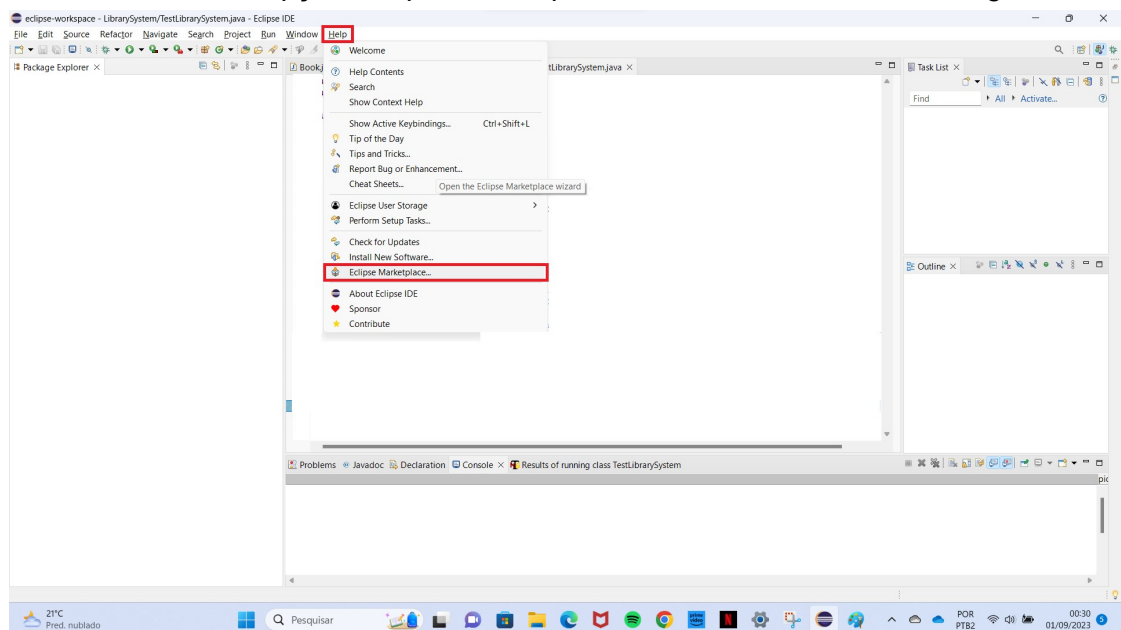


Figura 2 - Acesso ao Eclipse Marketplace

3º Passo: Quando aberto o TestNG, pesquisar na barra de pesquisa por TestNG e clicar em 'Install', conforme destacado na Figura 3.

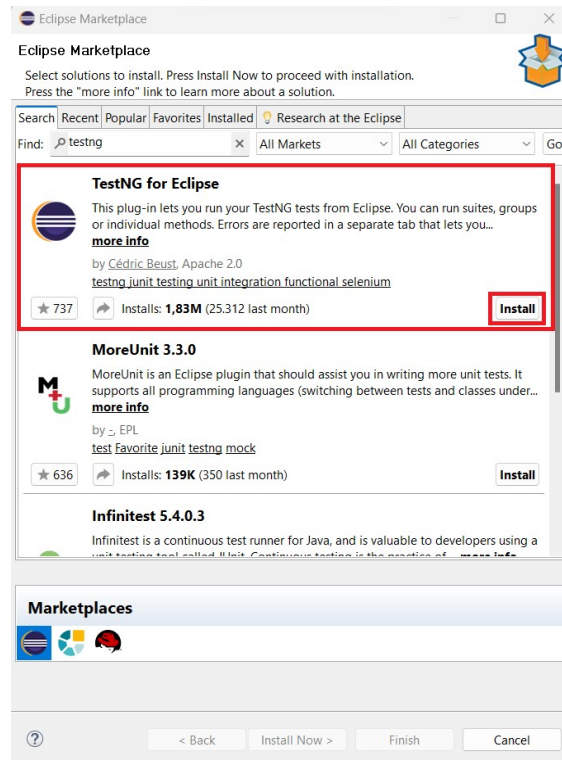


Figura 3 - TestNG no Eclipse Marketplace

4º Passo: Em seguida, deve-se selecionar as Features que devem ser instaladas e clicar em 'Finish', conforme a Figura 4. Logo após isso, a IDE reiniciará e o plugin estará instalado.

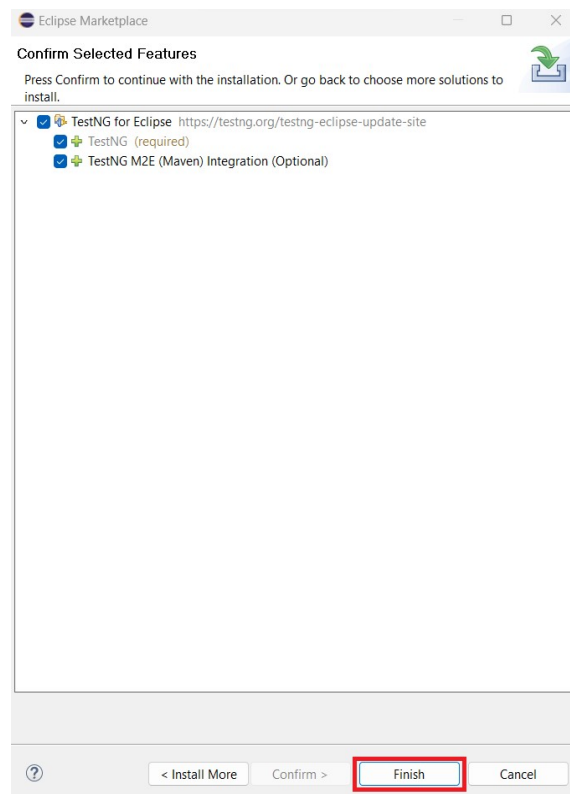


Figura 4 - Seleção das Features do TestNG a serem instaladas

5º Passo: Agora, com o framework instalado, podemos seguir para a criação do projeto. Primeiro, deve-se criar um novo projeto Java, preenchendo o campo de 'Project Name', clicar em 'Next' e logo após em 'Finish', conforme a Figura 5.

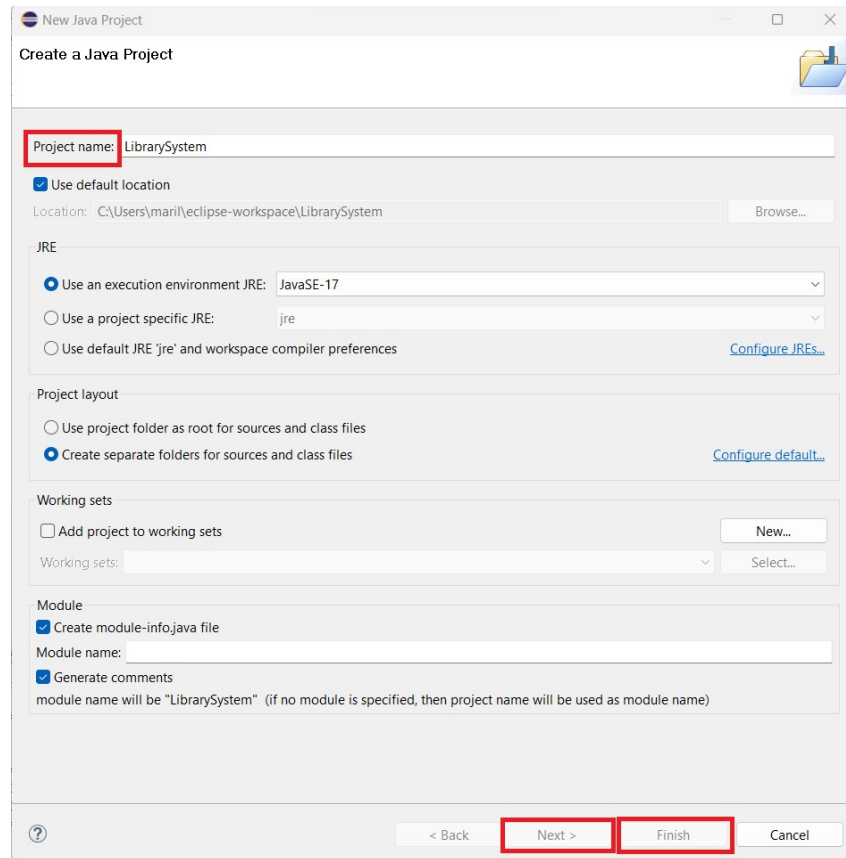


Figura 5 - Tela para a criação de um novo projeto Java

6º Passo: Depois da criação do projeto, deve-se adicionar o módulo de TestNG no módulo principal. Isso é feito clicando com o botão direito no nome do projeto Java e seguindo o caminho 'Build Path' e 'Configure Build Path'. Em seguida, deve-se ir para a aba 'Libraries', clicar em 'Add Libraries' e selecionar o TestNG, de modo que a configuração final fique como na Figura 6. Ao seguir essa etapa, se atentar para adicionar as extensões e arquivos JARs necessários para que o código rode sem problemas.

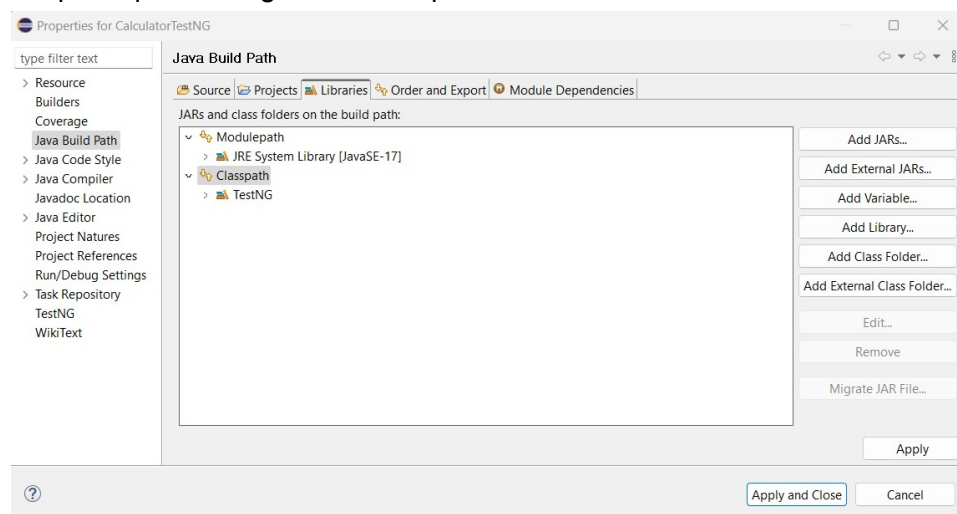


Figura 6 - Adição do TestNG no projeto Java

7º Passo: Agora, vamos começar de fato o projeto de aplicação do TestNG no código do sistema de cadastro de livros em uma biblioteca. O código é bem simples e está disponível no repositório <https://github.com/mcacs/testngproject>, e sua complexidade foi a melhor possível dado meu conhecimento na linguagem de programação e a dificuldade para mexer na ferramenta. Deve-se destacar que o foco principal desse tutorial é observar as características da ferramenta proposta. Para a criação do código, deve-se criar dois arquivos contendo as classes 'Book' (responsável por representar uma instância de um livro na biblioteca) e 'Library' (responsável por representar uma biblioteca com uma lista de livros) e um arquivo do tipo classe TestNG, clicando com o botão direito no pacote src e seguindo o caminho 'New' e 'Package' e, depois, 'New' e 'Class'. Um trecho do código dos testes é mostrado na Figura 7.

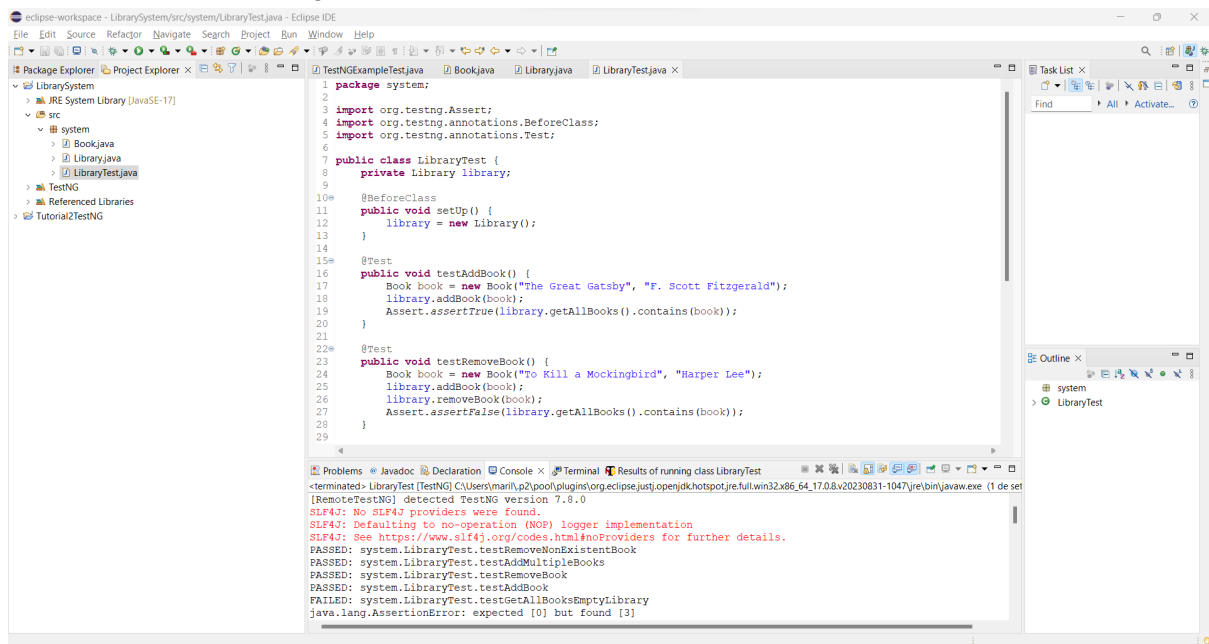


Figura 7 - Arquivo de teste do tipo TestNG contendo o código de teste do sistema de cadastro

Antes de passar para o próximo passo, é importante destacar o uso de Anotações no código de teste. São elas: `@BeforeClass`, que é usada para marcar um método que deve ser executado antes da execução de todos os outros métodos de teste na classe de teste; e `@Test`, que é usada para marcar um método como um método de teste. Existem várias outras marcações no TestNG e essa característica representa uma vantagem, como dito anteriormente.

8º Passo: Com o código escrito, o próximo passo é rodar o arquivo de teste para obter um pequeno *report* com o resultado dos testes rodados. Importante destacar que, para este caso, foram criados cinco tipos de teste. O primeiro, 'testAddBook', é responsável por verificar se um livro é corretamente adicionado à biblioteca; o segundo, 'testRemoveBook', é responsável por verificar se um livro é corretamente removido da biblioteca; o terceiro 'testGetAllBooksEmptyLibrary', verifica se a lista de livros está vazia quando a biblioteca é criada; o quarto, 'testAddMultipleBooks', verifica se é possível adicionar vários livros à biblioteca e se eles estão presentes na lista de livros; e, o quinto, 'testRemoveNonExistentBook', verifica se tentar remover um livro que não está na lista causa erros. O caminho para rodar o teste é clicar no botão direito em cima do arquivo de

teste e 'Run As' e 'TestNG File'. Os resultados gerados pelo Eclipse são mostrados na Figura 8. Importante destacar que ele informa quantos testes foram feitos, quais foram eles, quantos falharam e quantos foram bem sucedidos.

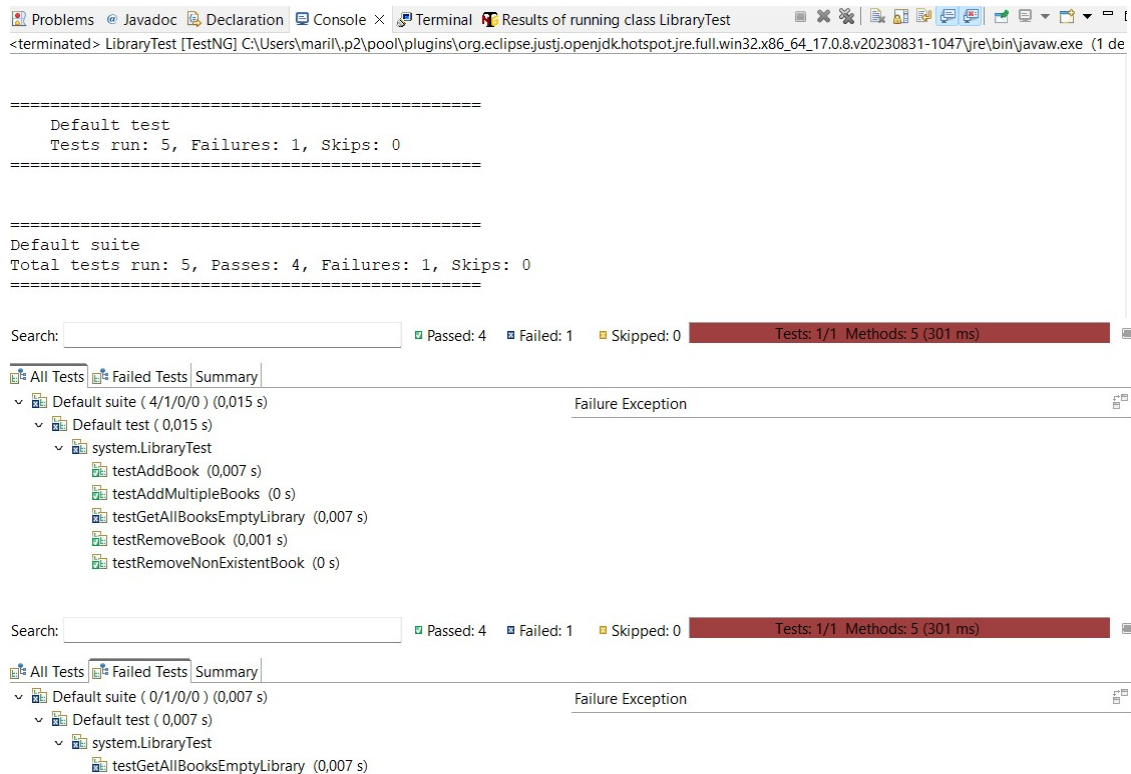


Figura 8 - Resultado de rodar testes automatizados com o TestNG no código

9º Passo: Por fim, após rodar o arquivo de teste, o próprio TextNG gera um arquivo de formato .xml com informações para definir suíte de testes, testes individuais, classes de teste, grupos de testes e várias opções de execução. Esse arquivo se localiza no diretório do projeto, geralmente sob o nome de 'testing_results.xml'. Na Figura 9 há um trecho do conteúdo do arquivo .xml.

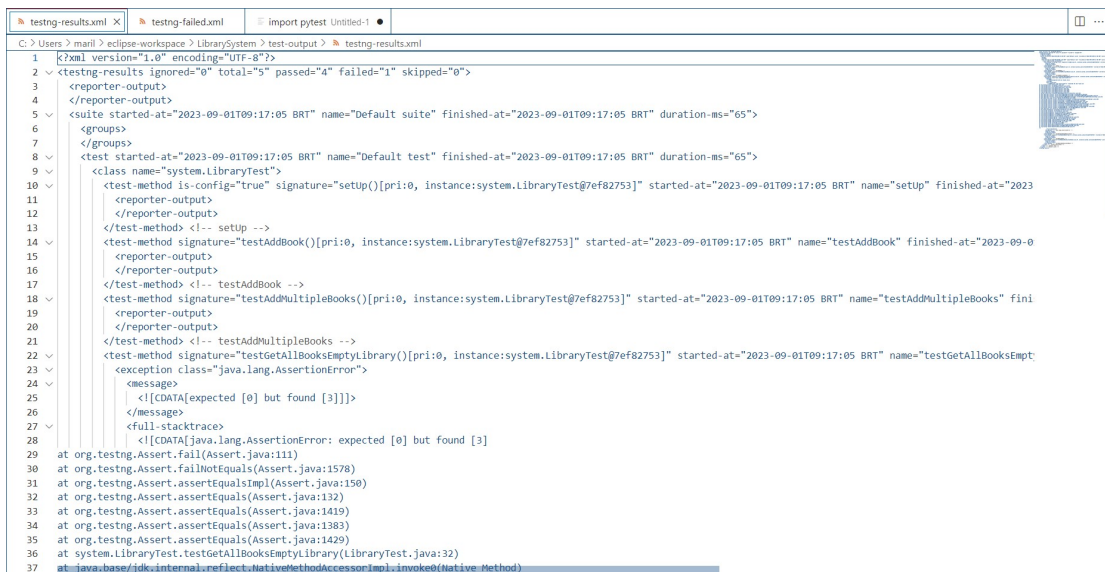
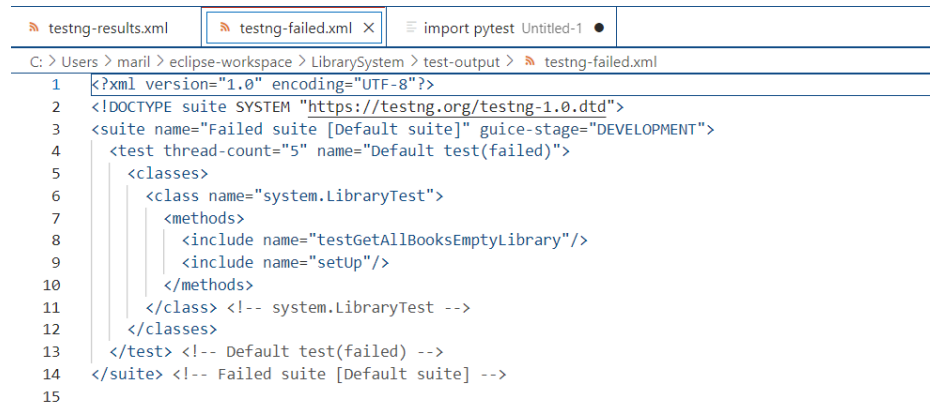


Figura 9 - Trecho do arquivo .xml gerado pelo TestNG

Como no código houve um teste que falhou, também é gerado um arquivo .xml contendo informações do teste que teve falha, geralmente com o nome de 'testing-failed.xml'. Nesse caso, o teste que verifica se a lista de livros está vazia quando a biblioteca é criada não foi bem sucedido. O código .xml para essa falha é mostrado na Figura 10.



The screenshot shows the Eclipse IDE interface. The top toolbar includes icons for 'testng-results.xml', 'testing-failed.xml' (which is selected), and 'import pytest'. The breadcrumb path is 'C: > Users > maril > eclipse-workspace > LibrarySystem > test-output > testing-failed.xml'. The main editor displays the XML content of the failed test report, with line numbers 1 through 15 on the left. The XML is a TestNG suite report for a failed test.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
3 <suite name="Failed suite [Default suite]" guice-stage="DEVELOPMENT">
4   <test thread-count="5" name="Default test(failed)">
5     <classes>
6       <class name="system.LibraryTest">
7         <methods>
8           <include name="testGetAllBooksEmptyLibrary"/>
9           <include name="setUp"/>
10        </methods>
11      </class> <!-- system.LibraryTest -->
12    </classes>
13  </test> <!-- Default test(failed) -->
14 </suite> <!-- Failed suite [Default suite] -->
15
```

Figura 10 - Código .xml do teste que falhou

Desafios

Antes de encerrar o documento, gostaria de ressaltar alguns aspectos que representaram desafios para mim, durante a execução desse documento:

- Minha falta de habilidade com Java, pois tenho conhecimento quase nulo na linguagem;
- A necessidade de baixar outra IDE para rodar a ferramenta;
- Além disso, há a necessidade de adicionar vários arquivos do tipo JAR à biblioteca de TestNG para que o arquivo de teste seja rodado de forma correta, o que não é bem esclarecido na maioria dos materiais disponíveis na internet.

Referências

[1] TESTNG Framework Tutorial: A Comprehensive Guide, with Examples and Best Practices. **LambdaTest**, 2023. Disponível em: <

<https://www.lambdatest.com/learning-hub/testng/>>. Acesso em: 30 de Agosto de 2023.

[2] TESTNG Tutorial: Introduction to TestNG Framework. **Software Testing Help**, 2023.

Disponível em: <<https://www.softwaretestinghelp.com/testng-tutorial/>>. Acesso em 31 de Agosto de 2023.

Link do vídeo do tutorial:

<https://drive.google.com/file/d/1Y8WLo1OaKrxQtJ8WxuKmaG1wbSgkceIE/view?usp=sharing>