Índice

Assuntos: Teste de Software - JUnit 5

Abertura Alunos:
Apresentação pitch
Leticia e Jaqueline

Teste de Software - JUnit 5

O que são testes unitários e qual a sua importância?

- O teste de software é uma forma de avaliar a qualidade da aplicação e reduzir os riscos de falhas no código ao ser colocado em operação.

O que é XP ?

XP (Extreme Programming) é uma metodologia ágil de desenvolvimento de software que enfatiza práticas como comunicação próxima com o cliente, feedback contínuo, simplicidade, feedback rápido, entre outras. No contexto de testes unitários, XP promove uma abordagem rigorosa e contínua para escrever testes unitários automatizados.

Nos princípios do XP, os testes unitários são uma parte crucial do processo de desenvolvimento de software, e são escritos antes ou simultaneamente à implementação do

código. Isso é conhecido como Test-Driven Development (TDD), onde os testes são escritos antes do código de produção e o código de produção é então desenvolvido para passar nesses testes.

Além disso, XP promove a execução frequente de todos os testes automatizados como parte do processo de construção do software, garantindo que quaisquer regressões sejam detectadas rapidamente. Isso ajuda a manter a qualidade do código e a fornecer um rápido feedback aos desenvolvedores sobre possíveis problemas. Em resumo, nos testes unitários, XP enfatiza a prática do TDD e a execução regular de todos os testes automatizados.

O que é o JUnit?

JUnit é uma biblioteca de teste para a linguagem de programação Java. Ela é um framework de teste. O JUnit oferece um conjunto de anotações e classes que permitem aos desenvolvedores escrever e executar testes automatizados para garantir a qualidade do código. Com o JUnit, os testes podem ser organizados, executados e os resultados podem ser analisados de forma eficiente.

O que é TDD?

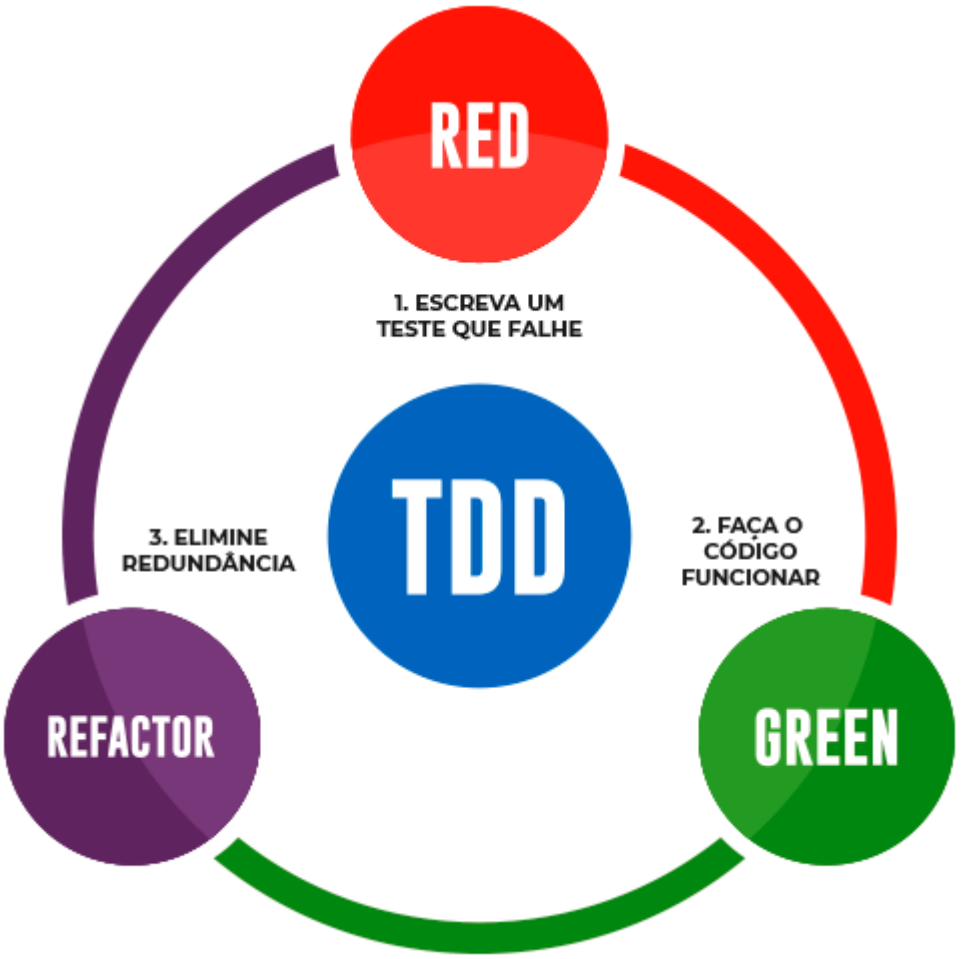
TDD, ou Test-Driven Development (Desenvolvimento Orientado a Testes), é uma abordagem de desenvolvimento de software onde os testes automatizados são escritos antes do código de produção. O processo segue um ciclo de desenvolvimento iterativo e incremental, composto por três etapas principais:

Escrever um teste: O desenvolvedor escreve um teste automatizado que define uma pequena unidade de funcionalidade desejada. Este teste normalmente falha inicialmente, pois o código de produção ainda não foi implementado para satisfazer a condição do teste.

Fazer o teste passar: O desenvolvedor, em seguida, implementa o código de produção mínimo necessário para fazer o teste passar. Isso geralmente significa escrever o código mais simples que pode satisfazer a condição do teste.

Refatorar o código: Uma vez que o teste esteja passando, o desenvolvedor pode refatorar o código, ou seja, reestruturá-lo para melhorar a sua qualidade, sem alterar o comportamento externo. Durante este processo, os testes garantem que o comportamento esperado do código permaneça inalterado.

Este ciclo é repetido continuamente, com testes adicionais sendo escritos para novas funcionalidades ou para garantir que as alterações não quebrem o código existente. O TDD promove a entrega de código de alta qualidade, pois os testes são escritos para validar o comportamento esperado do software em todas as etapas do desenvolvimento. Além disso, fornece um feedback rápido ao desenvolvedor, ajudando a detectar e corrigir problemas mais cedo no processo de desenvolvimento.



Testes Unitários :

Os testes unitários são práticas de desenvolvimento de software em que unidades individuais de código (como funções, métodos ou classes) são testadas de forma isolada para garantir que funcionem conforme o esperado.
Objetivos dos Testes Unitários:

Prevenção de bugs, criação de confiança no código e verificação de sucesso e falha são objetivos essenciais dos testes unitários.
Momento de Execução dos Testes:

Os testes unitários são executados no início da formulação do código e de maneira regular, muitas vezes diariamente, para identificar e corrigir bugs de maneira contínua e evitar a acumulação de problemas.
Isolamento de Partes do Código:

Os testes unitários isolam partes específicas do código para testar individualmente, permitindo a validação independente do comportamento de cada unidade.
Principais Regras para Gerar Testes:

A abordagem "Do simples ao complexo" é uma prática comum, onde começamos testando casos simples antes de avançar para casos mais complexos.
A decisão de não testar métodos triviais, como getters e setters, é uma prática comum, pois esses métodos geralmente têm um comportamento direto e não requerem testes extensivos. No entanto, há casos em que testar tais métodos pode ser benéfico, especialmente se houver lógica incorporada a eles.

PS : ***Ao escrever testes, sempre verifique se a importação dos pacotes do JUnit na Classe de testes estão corretos. O JUnit 5 tem como pacote base org.junit.jupiter.api.***

Spring Testing Annotations

O Spring Boot Testing trabalha de forma integrada com os principais Frameworks de Teste do Mercado tais como: **JUnit**, **MockMVC** (Parte integrante do Spring Boot Testing), entre outros. Para escrever os nossos testes utilizamos o **JUnit 5**.

@SpringBootTest : A anotação @SpringBootTest cria e inicializa o nosso ambiente de testes.

A opção **webEnvironment = WebEnvironment.RANDOM_PORT** garante que durante os testes o Spring não utilize a porta da aplicação (em ambiente local nossa porta padrão é a 8080), caso ela esteja em execução. Através da opção, o Spring procura uma porta livre para executar os testes**.**

@TestIstance indica que o ciclo de vida vai começar e terminar.

1) O **LifeCycle.PER_METHOD:** ciclo de vida padrão, onde para cada Método de teste é criada uma nova instância da Classe de teste. Quando utilizamos as anotações @BeforeEach e @AfterEach é necessário utilizar esta anotação.

2) O **LifeCycle.PER_CLASS**: uma única instância da Classe de teste é criada e reutilizada entre todos os Métodos de teste da Classe. Quando utilizamos as anotações @BeforeAll e @AfterAll é necessário utilizar esta anotação.

@Autowired vai injetar a dependência.

O @Autowired é uma anotação em Java utilizada em frameworks de injeção de dependência, como Spring, para automatizar o processo de injeção de dependências. Em termos simples, a injeção de dependência é um padrão de projeto onde as dependências de uma classe são passadas para ela de fora, em vez de serem criadas internamente.

Com o @Autowired, você pode marcar variáveis de instância em uma classe e o framework injetará automaticamente as instâncias necessárias dessas dependências quando a classe for inicializada. Isso simplifica o código, pois evita a necessidade de criar manualmente as instâncias de dependência, tornando o código mais flexível e facilitando a manutenção

@BeforeAll: indica que o Método deve ser executado uma única vez antes de todos os Métodos da Classe, para criar algumas pré-condições necessárias para todos os testes (criar objetos, por exemplo).

@Test indica que meu método vai executar um teste

@DisplayName só para printar uma msg na hora de testar**.**

```
46
47      @Test
48      @DisplayName("Cadastrar Um Usuário")
49      public void deveCriarUmUsuario() {
50
51          HttpEntity<Usuario> corpoRequisicao = new HttpEntity<Usuario>(new Usuario(0L,
52              "Paulo Antunes", "paulo_antunes@email.com.br", "13465278", "-"));
53
54          ResponseEntity<Usuario> corpoResposta = testRestTemplate
55              .exchange("/usuarios/cadastrar", HttpMethod.POST, corpoRequisicao, Usuario.class);
56
57          assertEquals(HttpStatus.CREATED, corpoResposta.getStatusCode());
58
59      }
60
```

Requisição HTTP será enviada através do Método exchange() da Classe TestRestTemplate e a Resposta da Requisição (Response) será recebida pelo objeto corpoResposta do tipo ResponseEntity. Para enviar a requisição, o será necessário passar 4 parâmetros:

- **A URI:** Endereço do endpoint (/usuarios/cadastrar);
- **O Método HTTP:** Neste exemplo o Método POST;
- **O Objeto HttpEntity:** Neste exemplo o objeto requisição, que contém o objeto da Classe Usuario;
- **O conteúdo esperado no Corpo da Resposta (Response Body):** Neste exemplo será do tipo Usuario (Usuario.class).

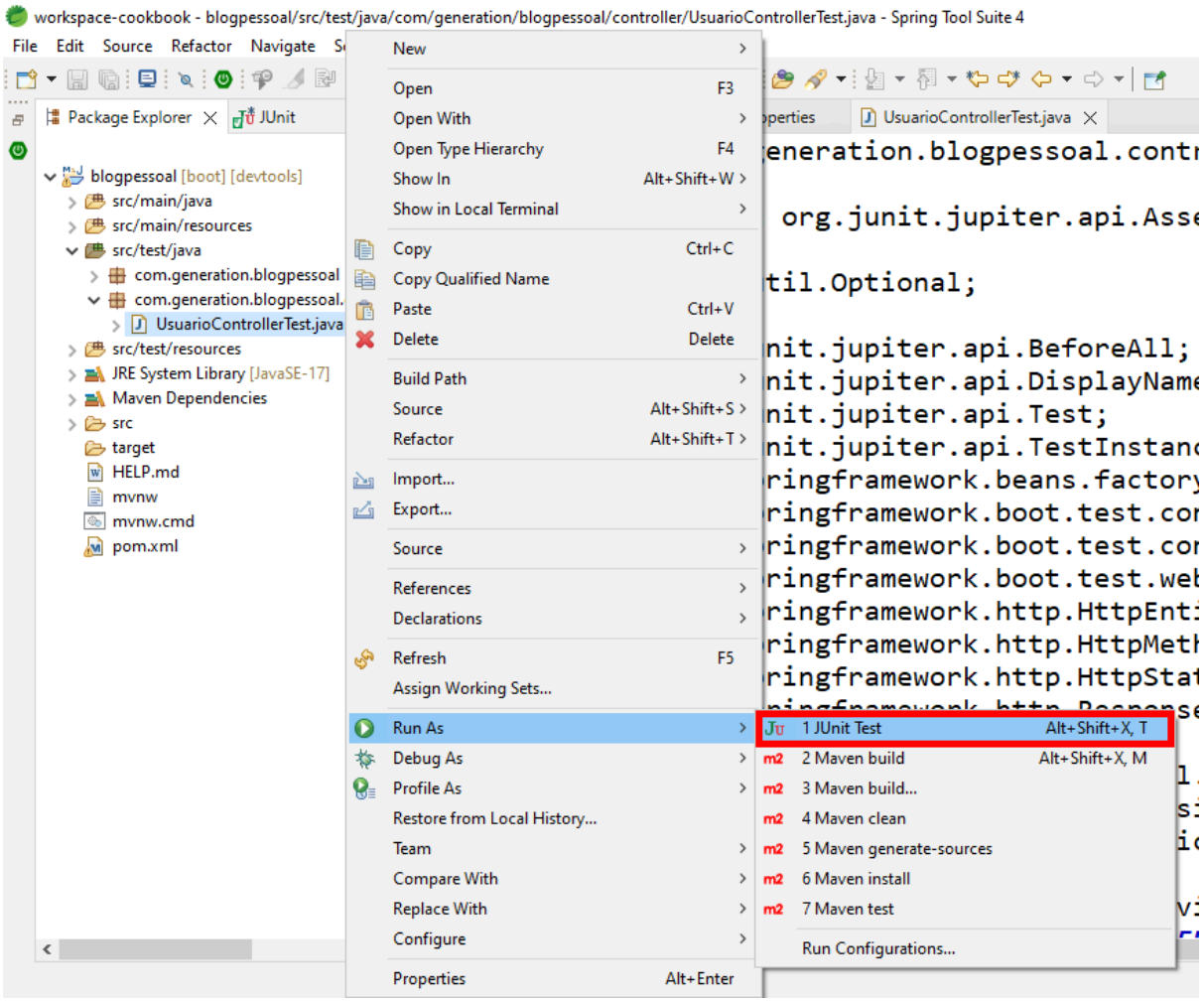
AssertEquals(), checamos se a resposta da requisição (Response), é a resposta esperada (CREATED 201). Para obter o status da resposta, vamos utilizar o Método getStatusCode() da Classe ResponseEntity.

Assertion	Descrição
assertEquals(expected value, actual value)	Afirma que dois valores são iguais.
assertTrue(boolean condition)	Afirma que uma condição é verdadeira.

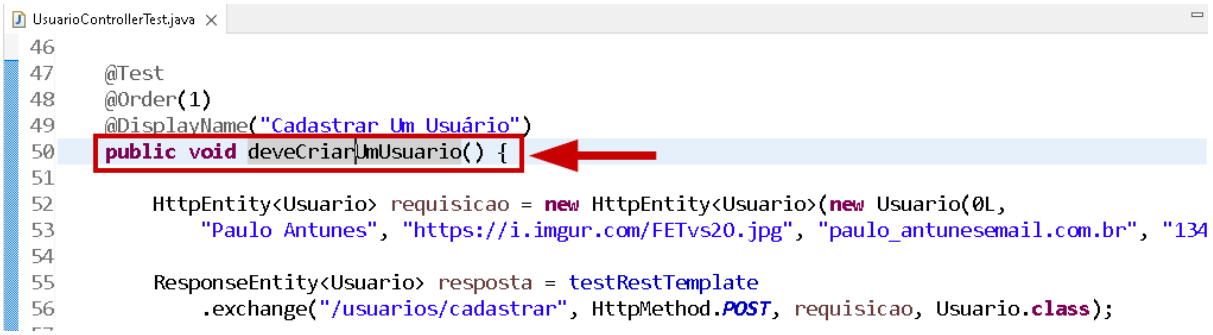
assertFalse(boolean condition)	Afirma que uma condição é falsa.
assertNotNull()	Afirma que um objeto não é nulo.
assertNull(Object object)	Afirma que um objeto é nulo.
assertSame(Object expected, Object actual)	Afirma que dois objetos referem-se ao mesmo objeto.
assertNotSame(Object expected, Object actual)	Afirma que dois objetos não se referem ao mesmo objeto.
assertArrayEquals(expectedArray, resultArray)	Afirma que array esperado e o array resultante são iguais.

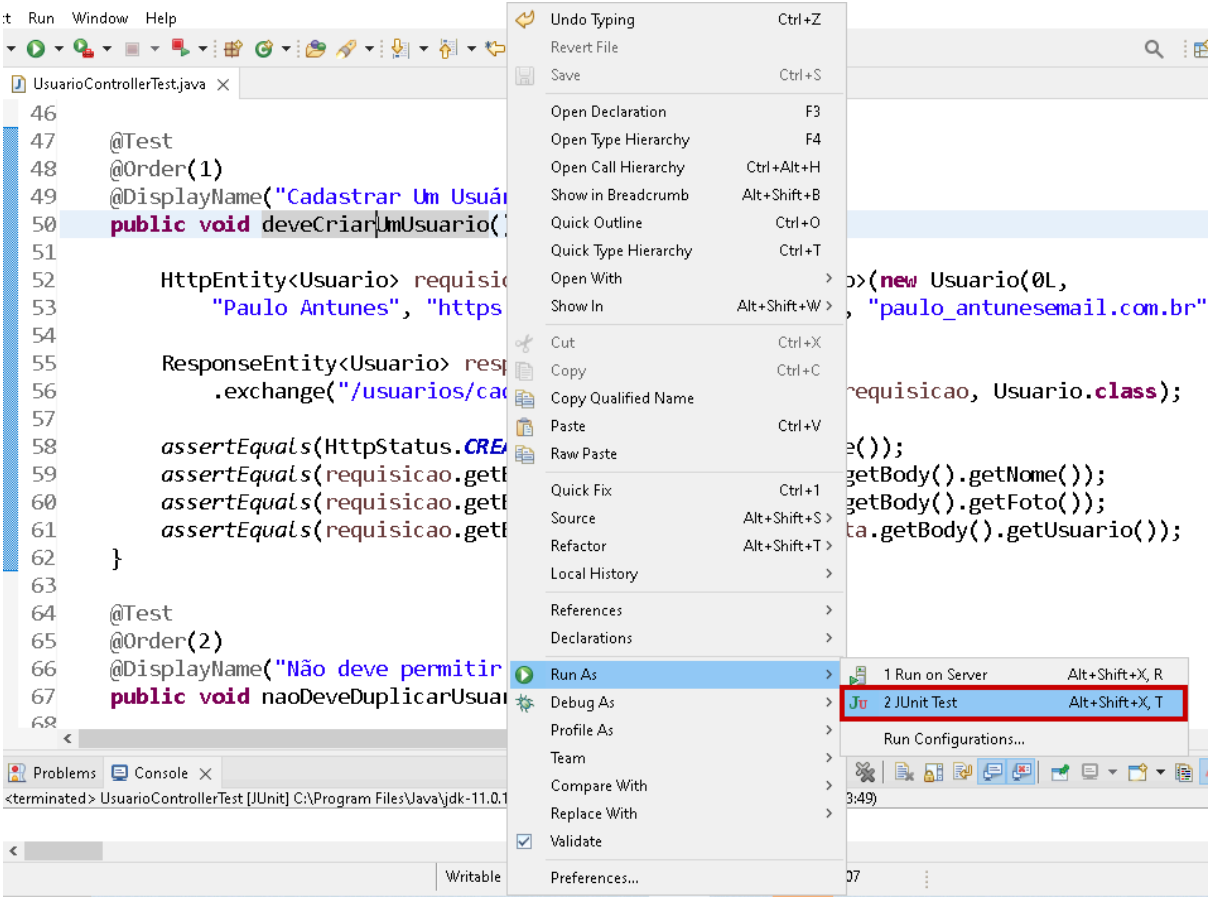
Para testar os Junits :

Pode ser a classe inteira :

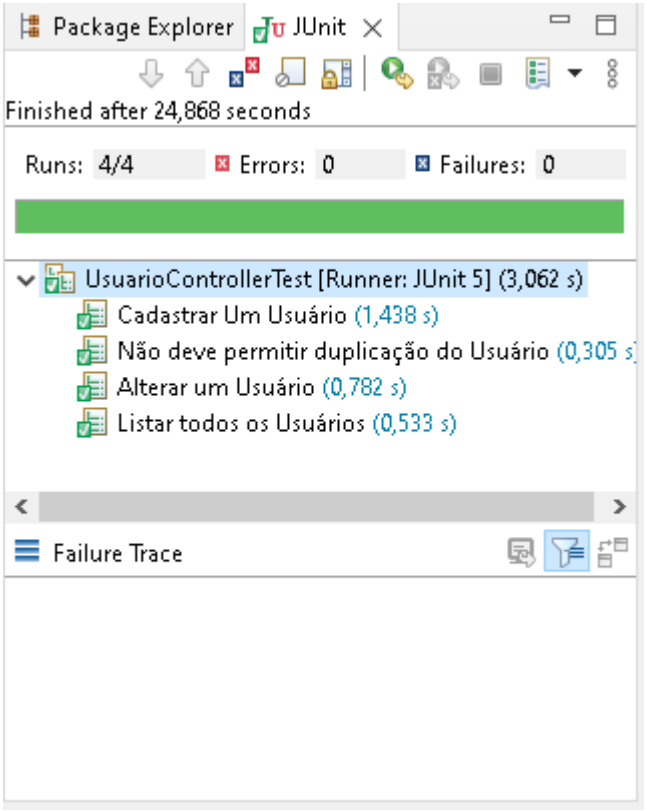


Por método é feito dentro do código:

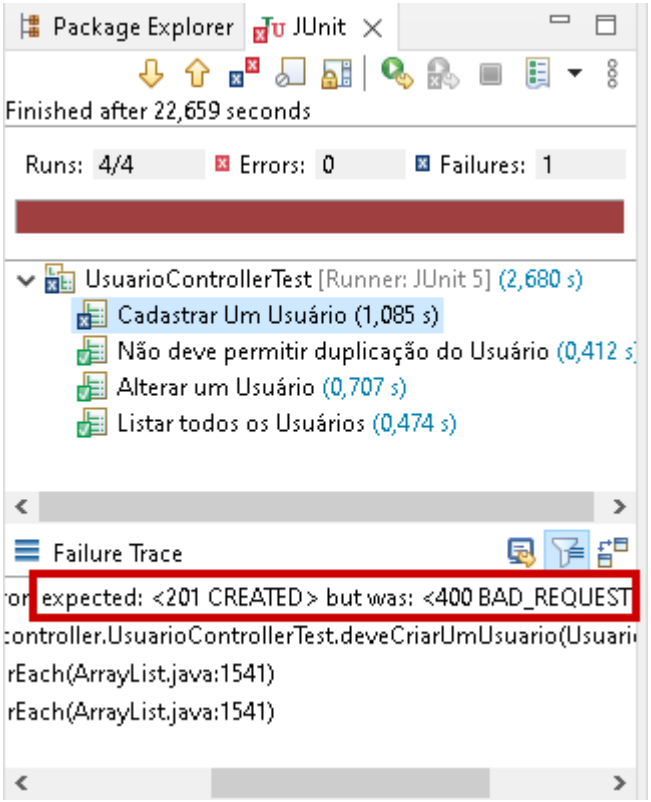




Rodando o teste você acompanha pela aba JUnit :



Caso de erro:



H2 banco de dados:

H2 é um banco de dados relacional escrito em Java. No contexto do Spring Boot, o H2 é frequentemente usado como um banco de dados em memória para desenvolvimento, teste e prototipagem de aplicativos. Ele é leve, rápido e pode ser facilmente configurado como um banco de dados embutido em uma aplicação Spring Boot.

O uso do H2 no Spring Boot permite que os desenvolvedores criem e testem rapidamente suas aplicações sem a necessidade de configurar um banco de dados separado. Além disso, como é um banco de dados em memória, os dados são perdidos quando a aplicação é reiniciada, o que pode ser útil para testes unitários e de integração isolados. No entanto, para ambientes de produção, geralmente é preferível usar um banco de dados mais robusto, como MySQL, PostgreSQL ou Oracle.

Lombok

Lombok é uma biblioteca para Java que oferece funcionalidades para reduzir a quantidade de código boilerplate (repetitivo) necessária em classes Java, como getters, setters, construtores, métodos equals() e hashCode(), entre outros. Ele permite aos desenvolvedores adicionar essas funcionalidades aos seus códigos Java usando anotações, reduzindo assim a verbosidade e tornando o código mais limpo e conciso.

Resumo :

JUnit: Framework de teste unitário para Java.
TDD (Test-Driven Development): Abordagem de desenvolvimento onde os testes são escritos antes do código de produção.
XP (Extreme Programming): Metodologia ágil que enfatiza práticas como TDD e comunicação próxima com o cliente.

✔ Boas Práticas :

- Faça testes pequenos.
- Faça testes rápidos: Os testes devem ser simples e objetivos porque serão executados o tempo todo.
- Faça testes determinísticos: O teste deve garantir o resultado.
- Faça testes independentes: Um teste não pode depender do resultado de outro teste.
- Utilize nomes auto descritivos: A ideia é que você entenda o que o teste faz sem precisar abri-lo.
- Insira poucas asserções em cada teste: O objetivo é que um teste seja responsável por apenas uma verificação.
- Sempre avalie os resultados dos seus testes.

Entrevistas

Tipos de entrevistas

Tipo de entrevista	Profissional que conduz	O que é avaliado
Técnica	Gerente da área ou profissional experiente (sênior)	Habilidades e conhecimentos técnicos da sua área e do cargo. Em algumas entrevistas o gestor que conduz pode inclusive trazer um estudo de caso relacionado a algum código, ou outro conhecimento específico da área a ser resolvido pelo candidato. Além disso, a entrevista técnica também serve, em alguns casos, para avaliar como o candidato conecta seu conhecimento técnico com a atividade da empresa.
Comportamental	Profissional de Recursos Humanos	Habilidades comportamentais e mentalidades. O profissional de Recursos Humanos avalia a adequação do candidato à cultura organizacional da empresa, a capacidade de comunicação clara, o entusiasmo e interesse do candidato pelo emprego, dentre outros.

Nota: Muitas vezes as entrevistas técnica e comportamental acontecem em uma só entrevista, onde o candidato é entrevistado por dois profissionais, um gestor da indústria e um profissional de Recursos Humanos.

Formato	Profissional que conduz	O que é avaliado
Em grupo	Geralmente é conduzida por um profissional de RH, mas também pode contar com um gestor da área	É um formato de entrevista usado principalmente quando há muitos candidatos aplicando para a vaga. Dentre os pontos avaliados estão: • Capacidade de trabalho em equipe. • Organização e gestão do tempo. • Comunicação, que inclui não só falar com clareza, mas saber ouvir e respeitar os demais candidatos. • Proatividade e capacidade de liderança, tanto de liderar um grupo como de ser liderado e também para se manter atualizado na indústria. • Conhecimentos técnicos e como usar seus conhecimentos para contribuir para o grupo. • Atenção e orientação ao detalhe.
Individual	Conduzida por um profissional de RH juntamente com um gestor da área	A entrevista individual é o momento onde o entrevistador pode conhecer melhor o candidato. Geralmente ocorre nas etapas finais de um processo seletivo, após as entrevistas em grupo. Dentre os pontos avaliados estão: • Habilidades comportamentais (comunicação, mentalidade de crescimento, orientação ao detalhe, etc). • Habilidades técnicas relacionadas à indústria e ao cargo

---

Links e afins

CookBook, JUnity 5 :

- MD 21
- MD 22
- MD 23

Lombok : https://projectlombok.org/

H2 banco de dados: H2

JUnit Documentação: https://junit.org/junit5/docs/current/user-guide/

Código Fonte:

https://github.com/conteudoGeneration/backend_blogpessoal_v3/tree/16_Testes_UsuarioController

SpringBoot Biblioteca:
https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.testing

---

💡 Dúvidas

- **Quando preciso fazer um teste unitário?**
  No início do desenvolvimento, já projetar as classes de testes em função das regras de negócio.

- **O que devo testar?**
  Ex: Se meu programa está criando os usuários corretamente, se meu programa está buscando as informações corretamente..
- **O que NÃO devo testar?**
  Métodos triviais, por exemplo: Getters e Setters.

- **Quais são as mensagens possíveis na execução do JUnit?**
  Sucesso(verde), Exceção(amarelo) ou Falha().