

Programação Orientada e Objetos II



Anhanguera

AVALIE
SUA PROFISSÃO

QUANDO APARECER EM SEU
PORTAL UMA AVALIAÇÃO SOBRE
SEU CURSO, RESPONDA:



NOTAS

9 ou 10

SIGNIFICA QUE VOCÊ INDICA

NOTAS

7 ou 8

SIGNIFICA QUE VOCÊ NÃO INDICA



Anhanguera



Anhanguera



Os testes em software são essenciais para que o produto entregue tenha os requisitos de qualidade imposto pelo cliente ou os cenários nos quais o sistema será implementado. Para avaliar esses testes, devemos primeiramente observar três conceitos básicos sobre testes (AMMANN, 2008):

1. Defeito: implementação incorreta feita em uma aplicação.
2. Erro: manifestação desse defeito ou falha.
3. Falha: comportamento externo que resulta em problemas de execução de um sistema (por exemplo, problemas de rede ou hardware).



Anhanguera

Dados esses cenários, podemos criar um grande arcabouço para fazer com que uma aplicação tenha seu número de defeitos minimizado e, assim, menor probabilidade de erros. Para isso, podemos executar diversas formas de testes obter um cenário com poucos erros. Também, é possível investir em uma documentação acurada e em um sistema de controle de versão de código, a fim de aumentar a confiabilidade do sistema e seu desenvolvimento. A forma inicial para a produção de testes são os testes unitários.



Anhanguera

Testes unitários em Java

Existem diversas categorias de testes, tais como de módulos, de integração, de sistema e de validação (SOMMERVILLE, 2016). Um dos testes iniciais que podem ser feitos é o unitário, cujo objetivo é verificar se os métodos estão funcionando. Uma das ferramentas utilizadas para realizar esse tipo de teste é o JUnit, com o qual é possível fazer diversas avaliações para confirmar se o código está funcionando corretamente (TAHCHIEV, 2010). O teste unitário pode ser comparado a testar e validar todas as peças de um carro antes de fazer a montagem. Se as peças estão funcionando corretamente, as chances de a montagem funcionar aumentam.



Anhanguera

Vamos utilizar um exemplo para demonstrar a criação de testes unitários. Considere a classe Calculadora criada no Quadro 3.2. Ela faz um conjunto de operações básicas, como somar (linhas 3 e 4), subtrair (linhas 5 e 6), dividir (linhas 7 e 8) e multiplicar (linhas 9 e 10). Com base nessa classe vamos desenvolver os testes unitários.



```
1. package U3S1;
2. public class Calculadora {
3.     public double somar(double a, double b)    {
4.         return a + b;
5.     }
6.     public double subtrair(double a, double b){
7.         return a - b;
8.     }
9.     public double dividir(double a, double b) {
10.        return a / b;
11.    }
12.    public double multiplicar(double a , double
13.    b) {
14.        return a * b;
15.    }
16. }
```



Anhanguera

Para executar esses testes, é necessário que alguma ferramenta auxilie no processo juntamente com o JUnit. O IDE Eclipse já possui integração para fazer os testes unitários, cuidando da sua criação e execução através de uma determinada classe. Ao utilizar o Eclipse, é preciso criar um JUnit Test Case, seguindo o menu File->New->JUnit Test Case. Na tela exibida, escolha uma pasta para armazenar os testes, um nome para o projeto e uma classe para testar no campo “Class under test”. Os demais parâmetros você pode deixar com os valores padrão.



O Quadro 3.3 apresenta um conjunto de testes elaborados para testar cada método da classe Calculadora. As linhas de 1 a 4 definem o pacote, as bibliotecas e a classe (isso é configurado de forma automática quando se cria a classe pelo Eclipse).

Dentro da classe CalculadoraTeste temos quatro métodos, cada um relacionado a um método da classe Calculadora. A marcação `@Test` se refere ao JUnit para afirmar que o método imediatamente abaixo é um teste. Utilizaremos como exemplo o teste entre as linhas de 6 a 9 chamado `void testSomar()`. Repare que nesse método, na linha 7, a classe Calculadora (`Calculadora c = new Calculadora();`) é instanciada e o método `double res = c.soma(10, 50);` é utilizado, guardando o resultado em uma variável. Por fim, na linha 8 é utilizado o `assertEquals(60, res);`. O `assert` vem do inglês afirmar ou confirmar, então, esse método vai comparar o 60 com o valor de `res`. Se esses valores forem iguais, é considerado como sucesso, mas caso seja diferente, significa que o teste falhou. Os outros métodos seguem a mesma lógica, fazendo a instância da classe que se quer testar e utilizando o `assert` para validar o resultado. Assim, a lógica dos métodos pode ser testada, evitando enviar para o cliente resultados inesperados.



```
1. package testes;
2. import static org.junit.jupiter.api.Assertions.*;
3. import org.junit.jupiter.api.Test;
4.
5. class CalculadoraTeste {
6.     @Test
7.     void testSomar() {
8.         Calculadora c = new Calculadora();
9.         double res = c.somar(10, 50);
10.        assertEquals(60, res);
11.    }
12. }
```

```
10. @Test
11.     void testSubtrair() {
12.         Calculadora c = new Calculadora();
13.         double res = c.subtrair(10, -15);
14.         assertEquals(25, res);
15.     }
16.     @Test
17.     void testDividir() {
18.         Calculadora c = new Calculadora();
19.         double res = c.dividir(10.0, 0);
20.         assertEquals(Double.isFinite(res),
21.             false);
22.     }
23.     @Test
24.     void testMultiplicar() {
25.         Calculadora c = new Calculadora();
26.         double res = c.multiplicar(120.115,
27.             12.5465);
28.         assertEquals(1507.01, 0.5, res);
29.     }
30. }
```



O Quadro 3.4 apresenta mais alguns exemplos de assert que podem ser utilizados.

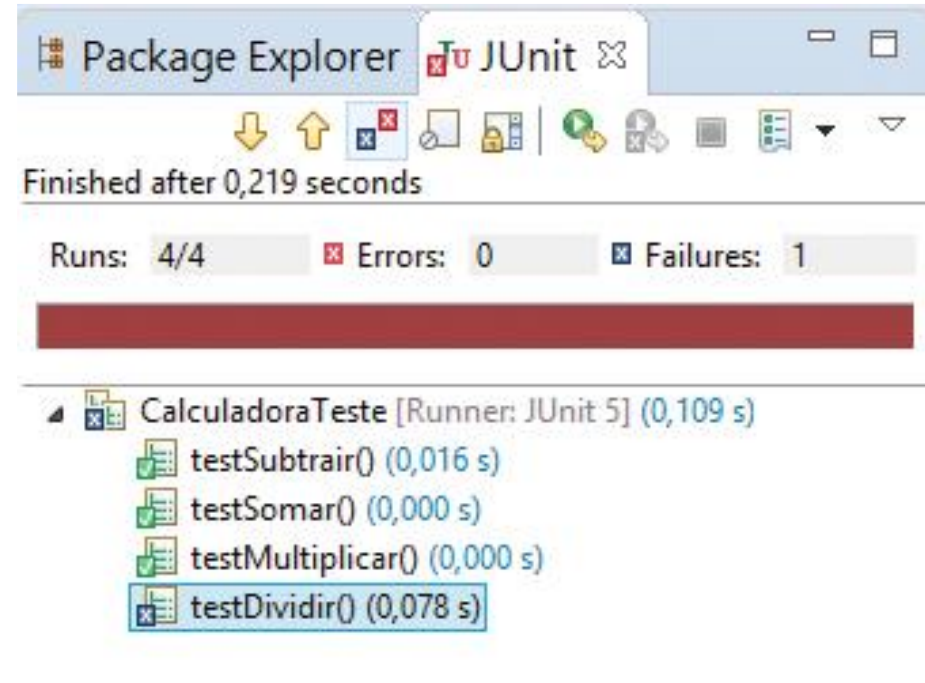
Tipo de assert:	Descrição:
<code>assertArrayEquals(float[] expected, float[] actual)</code>	Verifica se <code>float[] expected</code> é igual ao <code>float[] actual</code> , existem diversas sobrecargas para <code>double</code> , <code>int</code> e outros.
<code>assertNull(Object actual)</code>	Verifica se a instância passada é null.
<code>assertTrue(boolean condition)</code>	Verifica se a condição passa é true.



A Figura 3.1 apresenta o resultado do uso do Eclipse para essa classe de teste. O processo de execução dos testes é o mesmo que o de um software. Você pode clicar com o botão direito do mouse na classe de teste e escolher Run As, e o próprio Eclipse detecta que a classe é um teste, troca a interface utilizando os arcabouços do JUnit e exibe os resultados. Nesses testes é possível reparar que um deles (testDividir) falhou, sendo necessário validar se o código realmente possui um defeito ou se foi elaborado de forma incorreta. Nesse caso, o teste esperava que o resultado não fosse “Infinity”, todavia, o resultado de um número dividido por zero utilizando o tipo double é sempre “Infinity”. Dessa forma, o programador deve retornar à classe Calculadora e implementar uma forma de evitar divisão por zero.



Anhanguera





Anhanguera

Dessa forma, é possível verificar que os testes unitários são a primeira linha de defesa contra os bugs, o que ajuda a garantir que cada parte de um sistema esteja funcionando e, ainda que em caso de modificações, o sistema ainda mantenha seu funcionamento. Em certos cenários de desenvolvimento e algumas ferramentas não permitem que o software seja liberado ou integrado a uma aplicação sem que antes todos os testes unitários tenham sucesso. Outra forma de garantir que defeitos não sejam gerados em um sistema é a utilização correta da documentação de um sistema.



Anhanguera

Documentação de código fonte com o Javadoc

O Javadoc é uma ferramenta produzida pela própria Oracle que documenta todos os códigos do sistema implementado em Java. Com ela é possível fazer diversas marcações no decorrer do código para gerar uma versão em HTML para explicar o funcionamento de cada classe, método e outros de uma aplicação (DEITEL; DEITEL, 2016). Essa documentação tem o objetivo de explicar a função de cada elemento do código para os desenvolvedores, e não diretamente para o usuário (HORSTMANN, 2016). A documentação gerada por essa ferramenta é feita por meio de anotações específicas, utilizadas para identificar as partes do código e estruturar o documento HTML para que fique organizado. Algumas anotações usadas com frequência estão especificadas no Quadro 3.5.



Marcação	Descrição
<code>@author</code>	Define o autor da classe.
<code>@see</code>	Classes correlatas à que está sendo desenvolvida.
<code>@since</code>	Versão da classe.
<code>@param</code>	Parâmetro utilizado no método.
<code>@return</code>	Descrição do retorno do método.
<code>@code</code>	Altera a fonte do texto para indicar que é um código Java.

O Quadro 3.6 apresenta a classe Calculadora que foi alterada para ser documentada com o Javadoc.


```
/**
 * A classe calculadora é responsável por executar as
 * operações matemáticas básicas
 * @author Fabio Andrijauskas
 * @see java.lang.Math
 * @since 1.0
 */
public class Calculadora {
    /**
     * O método {@code somar} faz a soma de dois nú-
    meros
     *
     * @param a valor do primeiro número a ser soma-
    do.
     * @param b valor do segundo número a ser soma-
    do.
     * @return valor da soma de a e b
     * @see Math
     */
    public double somar(double a, double b) {
        return a + b;
    }
    /**
     * O método {@code subtrair} faz a subtração de
    dois números
     *
     * @param a valor do primeiro número a ser sub-
    traído.
     * @param b valor do segundo número a ser sub-
    traído.
     * @return valor da subtração de a e b
     * @see Math
     */
    public double subtrair(double a, double b){
        return a - b;
    }
}
```



Anhanguera

```
* O método ({@code dividir}) faz a divisão de
dois números
*
* @param a valor do primeiro número a ser divi-
dido.
* @param b valor do segundo número a ser divi-
dido.
* @return valor da divisão de a e b
* @see Math
*/
public double dividir(double a, double b) {
    return a / b;
}
/**
 * O método ({@code multiplicar}) faz a multipli-
cação de dois números
 *
 * @param a valor do primeiro número a ser mul-
tiplicado.
 * @param b valor do segundo número a ser multi-
plicado.
 * @return valor da multiplicação de a e b
 * @see Math
 */
public double multiplicar(double a , double b) {
    return a * b;
}
}
```



Anhanguera



Anhanguera

O Eclipse permite gerar toda a documentação com o Javadoc utilizando a opção Project > Generate Javadoc. A Figura 3.2 apresenta uma porção da documentação gerada da classe Calculadora. É possível verificar que ele utiliza a mesma formatação da documentação do Java, pois usa a mesma ferramenta. A documentação é criada na pasta do projeto de origem.



Anhanguera

PACKAGE **CLASS** USE TREE DEPRECATED INDEX HELP

PREV CLASS **NEXT CLASS** FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class Calculadora

java.lang.Object
Calculadora

```
public class Calculadora  
extends java.lang.Object
```

A classe calculadora é responsável por executar as operações matemáticas básicas

Since:

1.0

Author:

Fabio Andrijauskas

See Also:

Math

Constructor Summary

Constructors

Constructor and Description

calculadora()




A junção dos testes unitários e o mecanismo de documentação são um grande avanço na qualidade do desenvolvimento. Além desses dois elementos, é de grande importância uma forma de gerenciar as versões de um software e as contribuições de diversas pessoas da equipe.

Sistema de controle de versão

Existem diversos sistemas de controle de versão, como o Concurrent Versions System (CVS), Apache Subversion (SVN), git e outros. Um que está em grande destaque é o git, um sistema de controle de versão muito simples e versátil e, para utilizá-lo, é possível fazer a integração via Eclipse ou usar diretamente no terminal (CHACON, 2018). Faremos com o Eclipse, pois propicia uma interface gráfica. O primeiro passo consiste em instalar o plugin do git no eclipse, em Help > Install new software. Então, deve-se adicionar um novo repositório (botão Add ao lado do Manage) e utilizar o link do plugin, disponível em: <http://bit.ly/2MG17oX> (acesso em: 20 ago. 2018), conforme Figura 3.3.




Anhanguera

 Add Repository ×

Name:

Location:

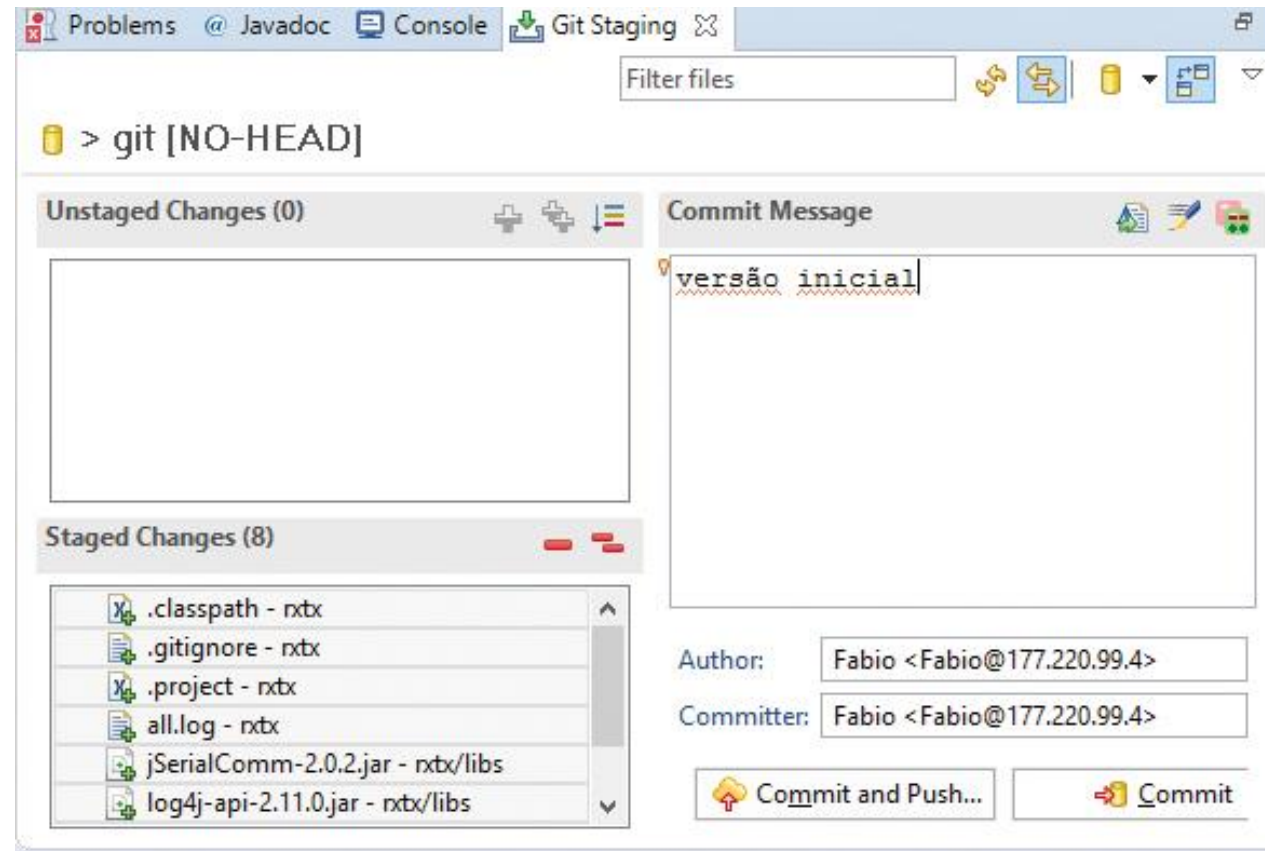




Após esse passo, é possível selecionar o egit em Work with. Deixe o restante de forma padrão, selecionando todos os componentes. Depois da instalação, é necessário criar um repositório git local. Ao clicar com o botão direito do mouse sob o projeto no Package Explorer, é possível selecionar a opção Team > Share Project. Na tela exibida, clique em Create para criar um repositório local (não pode ser no mesmo local do projeto). Com isso, o projeto estará com sistema de controle de versão git. Em seguida, é necessário fazer o primeiro push (envio dos arquivos) e commit (gravação das modificações do repositório). Para isso, é necessário clicar em Team > Commit para ver a interface da Figura 3.4. Nela você deve selecionar todos os arquivos e clicar no botão . Em seguida, no campo Commit Message, crie uma mensagem inicial e, por fim, utilize a opção Commit and push. Com isso, é possível configurar um servidor remoto para fazer a sincronização dos dados. Existem diversos serviços que proveem um repositório git, por exemplo o github.



Anhanguera





A criação da conta do repositório é feita no endereço <[https:// github.com](https://github.com)> (acesso em: 20 ago. 2018). Uma URL é gerada e deve ser utilizada na etapa de configuração do Eclipse após o Commit and push, conforme a Figura 3.4.

Com isso, todas as modificações feitas no código devem ser salvas localmente e enviadas clicando com o botão direito do mouse no projeto em Team > Commit and push.

Com essa etapa dos estudos é possível avançar nas formas de produção, controle e documentação, além de produzir código profissionais e duradouros.



Existem diversos tipos de testes que podem ser elaborados para verificar e validar um software. Alguns deles não necessitam de código-fonte para serem feitos e são elaborados utilizando a mesma interface que um usuário usaria. Porém, há testes que necessitam do código-fonte para serem elaborados, e um deles é o teste unitário.

Qual é a função principal de um teste unitário utilizando JUnit?

- a) Controlar a versão dos testes do sistema.
- b) Verificar o funcionamento da classe como um todo.
- c) Verificar o funcionamento dos métodos de uma classe.
- d) Testar o sistema de controle de documentação.
- e) Validar o sistema e seus módulos.



A documentação de um sistema é um elemento de grande importância para que seja possível explicar o que e como uma aplicação foi desenvolvida e quais são as formas de uso de um método, informando o objetivo da classe e os métodos, além de descrever os parâmetros necessários para utilizar os métodos apresentados. Para o Java, é possível utilizar o Javadoc.

Assinale a opção que possui apenas as marcações do Javadoc.

- a) @author, @see, @since, @param e @return.
- b) @option, @see, @control, @param e @return.
- c) !author, !see, !since, !param e !return.
- d) author, see, since, param e return.
- e) @attr, @see, @since, @param e @opt.