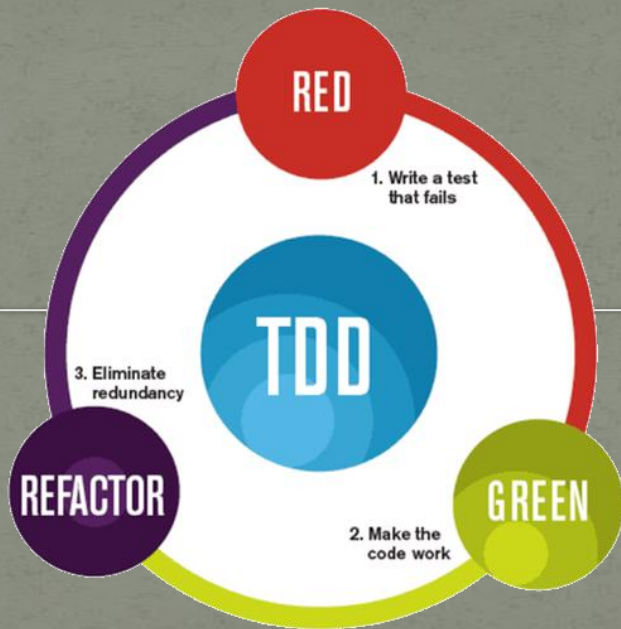


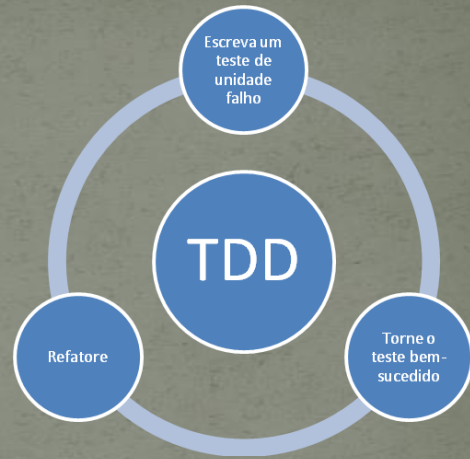
Test Driven Development



• Definição

• Definição

Test Driven Development (TDD) ou em português Desenvolvimento guiado por testes é uma técnica de desenvolvimento de software que se relaciona com o conceito de verificação e validação e se baseia em um ciclo curto de repetições: Primeiramente o desenvolvedor escreve um caso de teste automatizado que define uma melhoria desejada ou uma nova funcionalidade.



• História

Kent Beck, considerado o criador ou o *descobridor* da técnica, declarou em 2003 que TDD encoraja designs de código simples e inspira confiança. Desenvolvimento dirigido por testes é relacionado a conceitos de programação de *Extreme Programming*.



(Criador da *extreme programming*)

CARACTERÍSTICAS

- Características

Desenvolvimento dirigido por testes (TDD) requer dos desenvolvedores criar testes automatizados que definam requisitos em código antes de escrever o código da aplicação. Ou seja, o desenvolvedor deverá primariamente, compreender com detalhes as especificações do sistema e as regras de negócio.

Os testes são utilizados para facilitar no entendimento do projeto, segundo Freeman os testes são usados para clarear a ideia em relação ao que se deseja em relação ao código.

Modelo F.I.R.S.T.

Metodologias ágeis adotam a técnica “**First**”:

- **F** (Fast) - Rápidos: devem ser rápidos, pois testam apenas uma unidade;
- **I** (Isolated) - Testes unitários são isolados, testando individualmente as unidades e não sua integração;
- **R** (Repeatable) - Repetição nos testes, com resultados de comportamento constante;
- **S** (Self-verifying) - A auto verificação deve verificar se passou ou se deu como falha o teste;
- **T** (Timely) - O teste deve ser oportuno, sendo um teste por unidade.

Resumidamente, Primeiro escreva um teste que falhe, Depois escreva um código que faça o teste passar, então, Melhore o código escrito.

O TDD vai além do simples ato de testar, os testes passam a ser sua especificação, passam a ser a forma de você medir se o seu software está sendo conduzido para o real objetivo ou não. Literalmente, os **testes guiam o seu desenvolvimento**

Benefícios

Benefícios

- × Evitar passar horas apenas escrevendo testes.

- “Acabei, só faltam os testes.”

- Testes podem e devem servir de especificação.

O primeiro passo do TDD é justamente anotar os testes que serão necessários. Essa etapa ajuda a compreender melhor o que deve ser feito e escrever melhores especificações.

Benefícios

- × Planejar melhor a implementação
- × Escrever código mais testável
- × Maior produtividade pelo foco na resolução de problemas;
- × código limpo e bem escrito, resultado da simplicidade na hora de criá-lo e o tempo para refatorar;

Benefícios

- ✕ Facilidade e segurança para corrigir bugs, já que você trabalha com o código fração por fração;
- ✕ Modularidade e flexibilidade no seu código, proporcionados por essa quebra em pequenos objetivos;

Benefícios

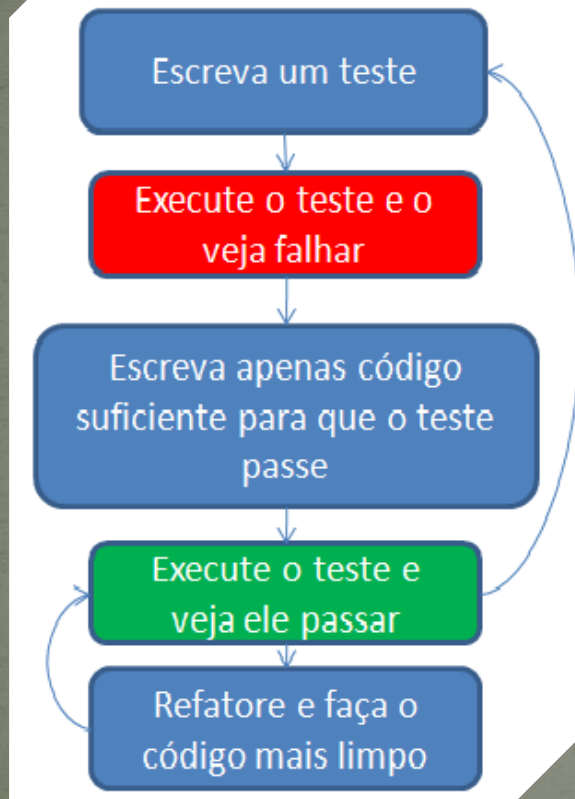
Resumindo tudo isso, o TDD é um conceito de programação de organizar melhor o seu trabalho e permite que você ofereça produtos com muito mais qualidade em muito menos tempo.

TDD

- Funcionamento

Basicamente o TDD se baseia em pequenos ciclos de repetições, onde para cada funcionalidade, um teste é criado antes. Estes ciclos de repetições se divide primariamente em 3 partes, em **RED**, **GREEN** e **REFACTOR**.

TDD



O **FUNCIONAMENTO** da técnica ágil TDD é muito simples, ao lado temos uma sequência do ciclo de funcionamento, analisaremos cada uma delas.

TDD

Escreva um teste



O primeiro passo é criar /codificar um teste que falhe, é pra falhar mesmo.

TDD

Escreva um teste

Execute o teste e o
veja falhar

Teste falhando

No próximo passo, executamos o teste e acompanhamos a falha, a falha ocorre porque não temos nenhuma funcionalidade implementada ainda.

TDD

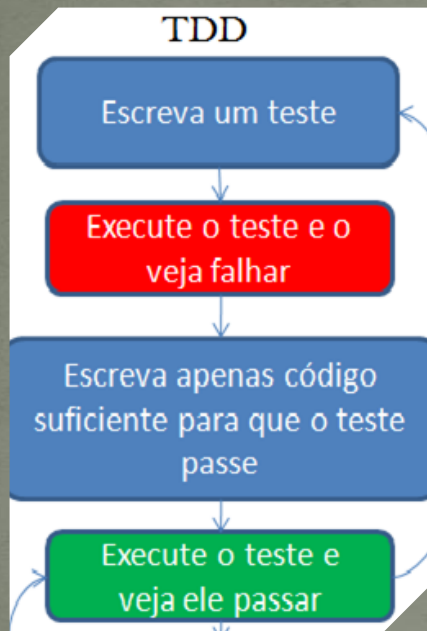
Escreva um teste

Execute o teste e o
veja falhar

Escreva apenas código
suficiente para que o teste
passe

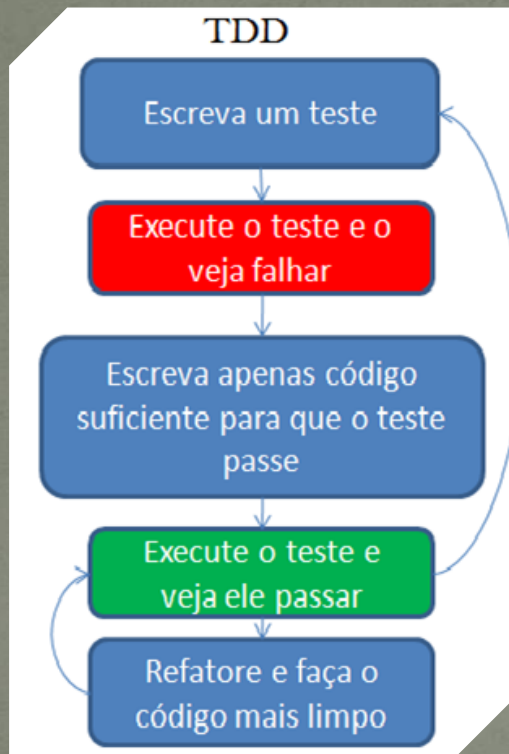
Nova funcionalidade

- Esse Código deve ser o mais simples possível.



Teste passando

Neste passo, você visualizará a funcionalidade passando no teste, retornando o resultado esperado.

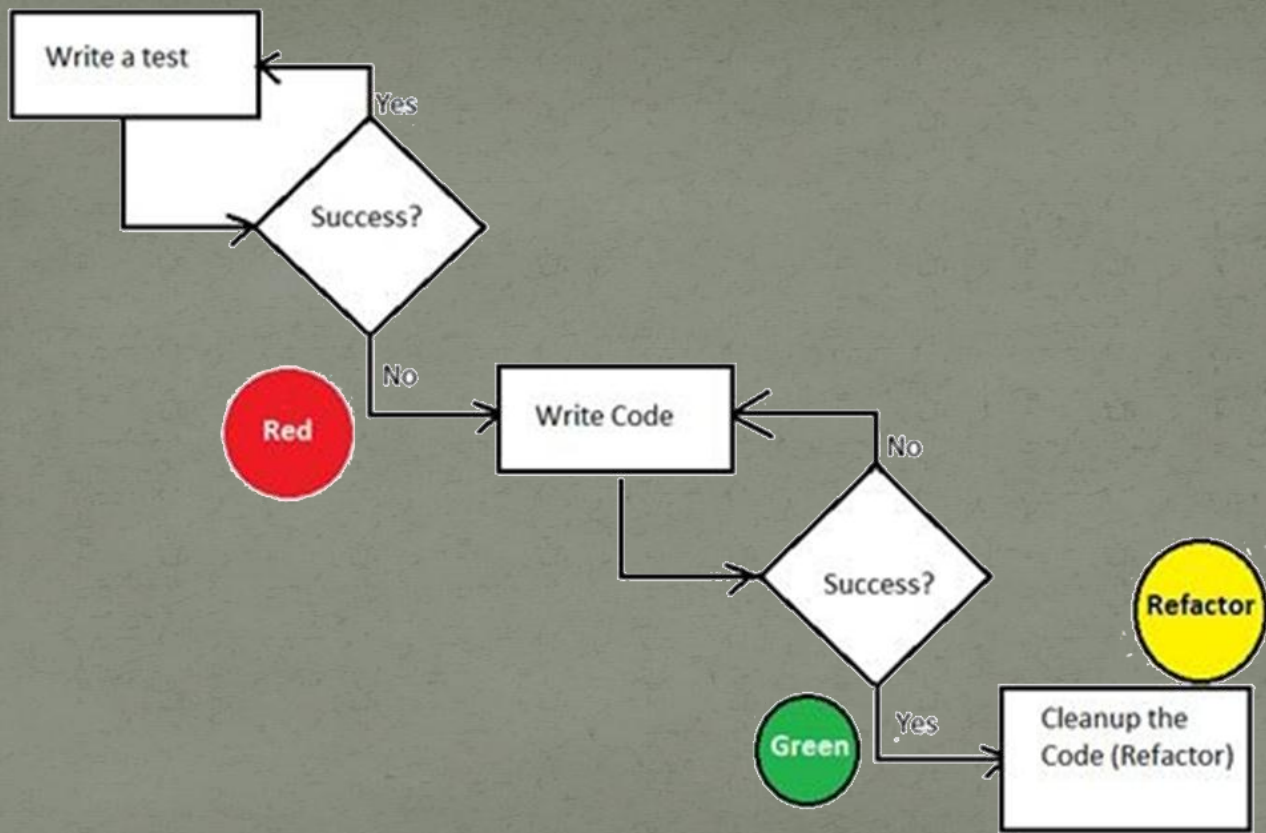


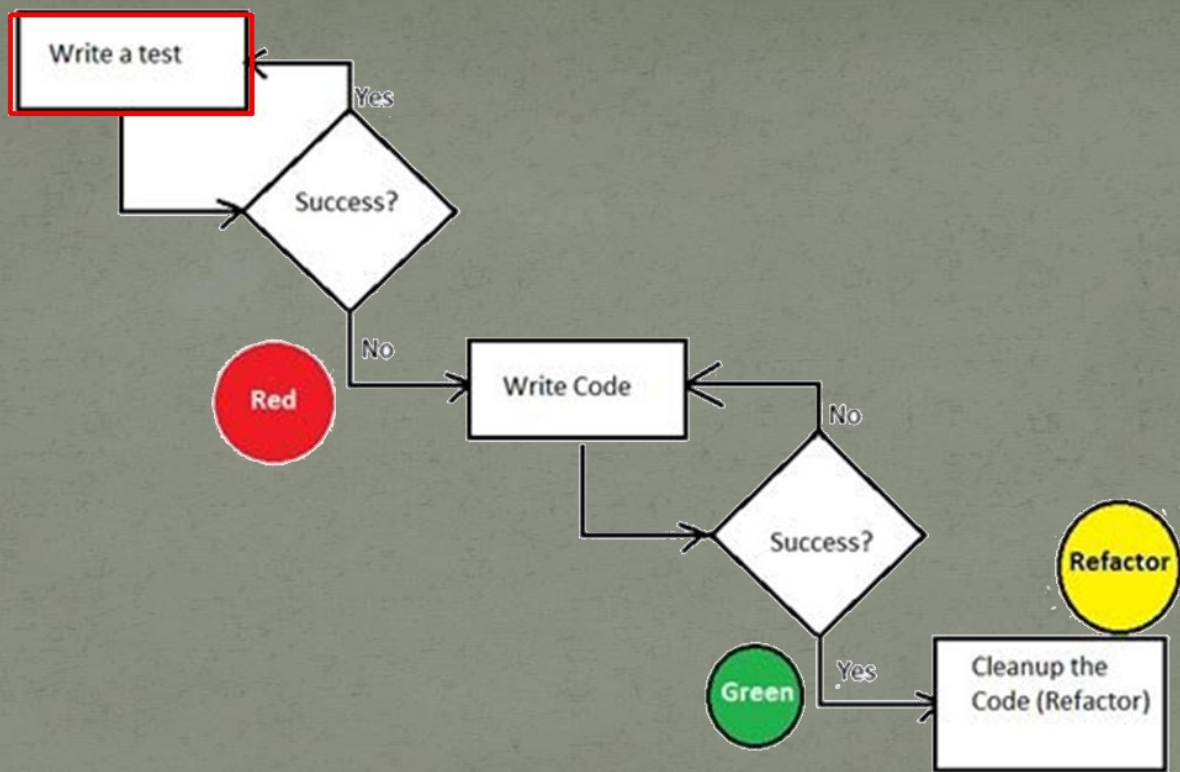
Refatoração

Código mais limpo significa: retirar duplicidades, renomear variáveis, extraír métodos, extraír Classes, extraír Interfaces, enfim deixando nosso Código simples e claro, e mais do que isso, > **FUNCIONAL.**

- Para a realização dos testes em TDD, contamos com uma infinidade de ferramentas, inclusive voltada para cada linguagem de programação, abaixo algumas delas.
- - PHPUnit – PHP
- - Cucumber – BDD
- - Jasmine e Mocha – node.js (Java Script)
- - JUnit – Java
- - PyUnit – Python
- - NUnit - dotNet

Exemplo prático





Write a test

Criando Classe para armazenar os testes

Success?



```
CalculadoraTest.java
1 package br.metodista.ead.ads1;
2
3 import org.junit.Test;
4 import static org.junit.Assert.*;
5
6 /**
7  *
8  * @author Rafael Guimarães Sakurai
9  */
10 public class CalculadoraTest {
11
12
13
14 }
15
```

Write a test

Criando Classe para armazenar os testes

Yes

Success?

```
CalculadoraTest.java
1 package br.metodista.ead.ads1;
2
3 import org.junit.Test;
4 import static org.junit.Assert.*;
5
6 /**
7  *
8  * @author Rafael Guimarães Sakurai
9  */
10 public class CalculadoraTest {
11
12     @Test
13     public void testarSoma() {
14         assertEquals(2, 2);
15     }
16
17 }
18
```


Write a test

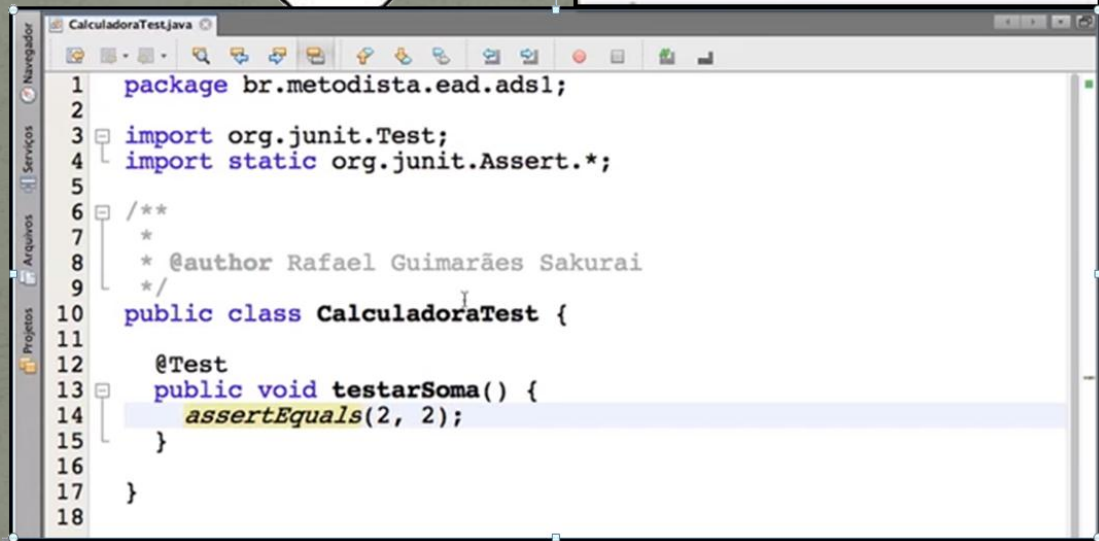
Yes

Success?

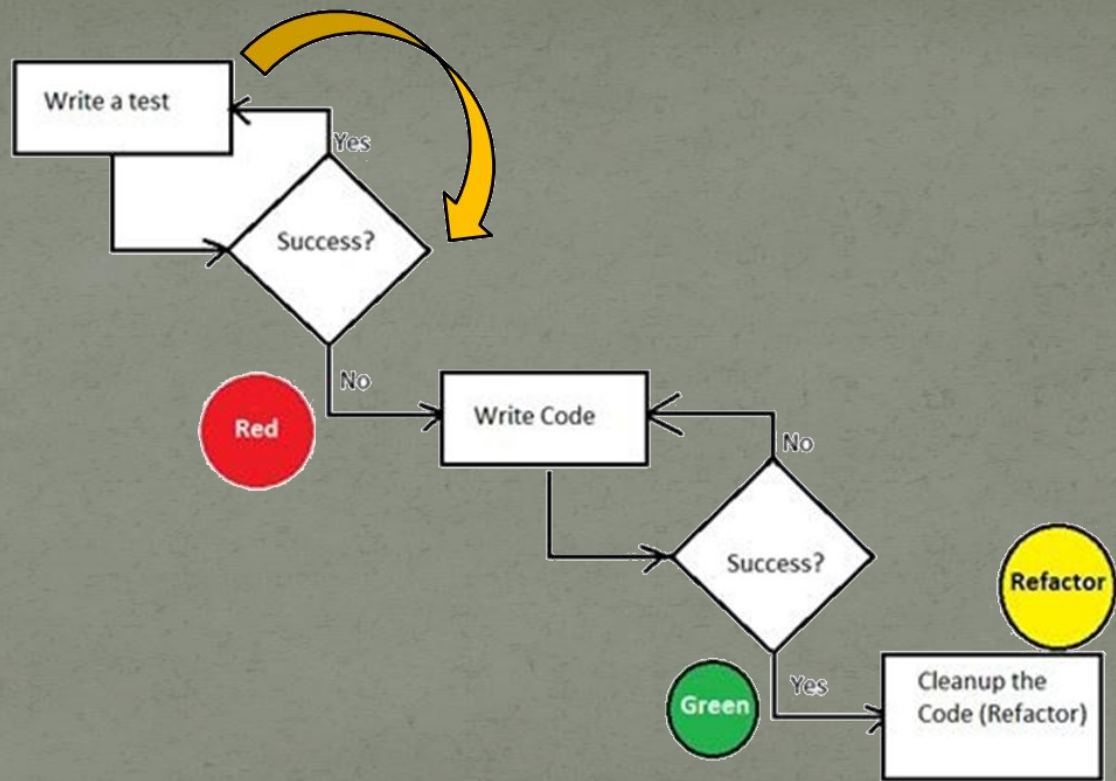
org.junit.Assert

```
public static void  
assertArrayEquals(Object[]  
expecteds, Object[] actuals)
```

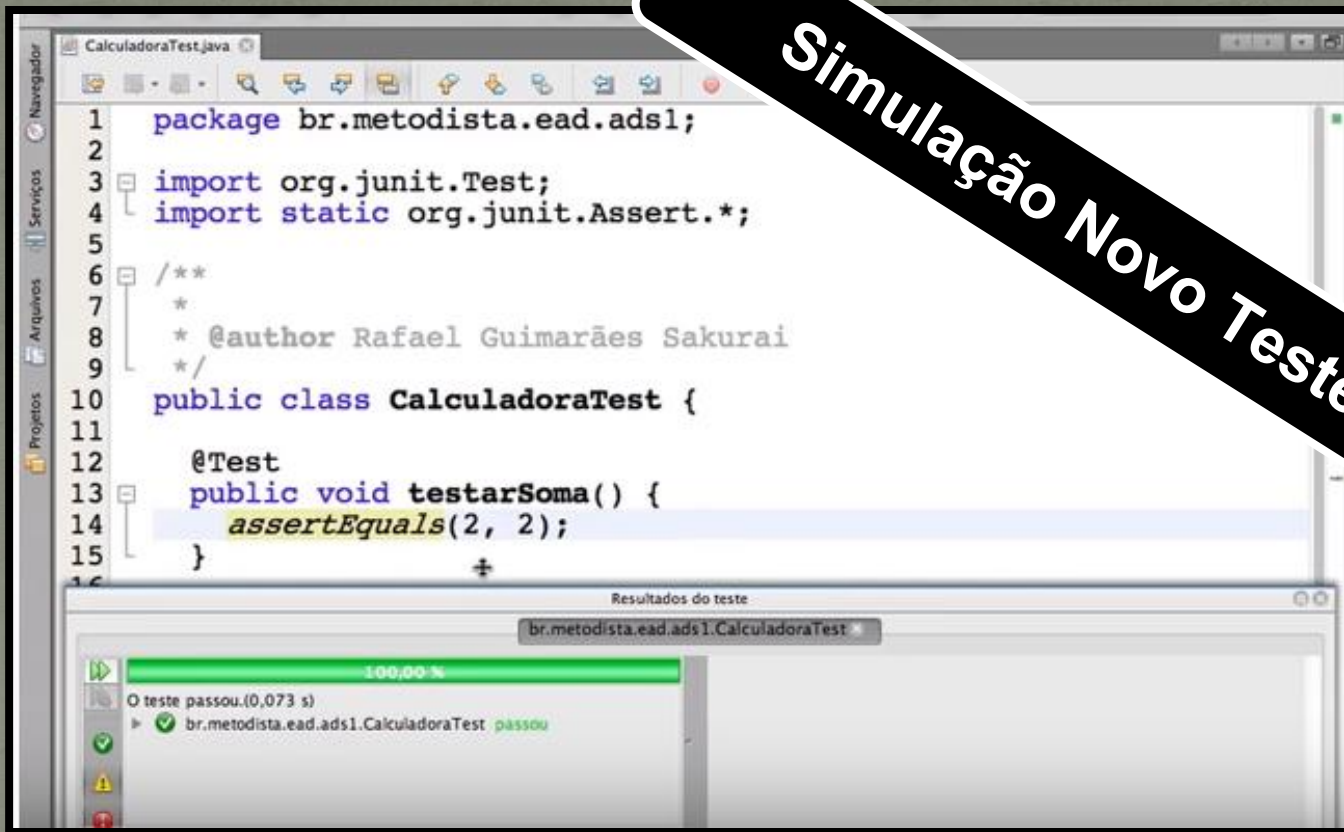
Asserts that two object arrays are equal. If they are not, an `AssertionError` is thrown. If `expected` and `actual` are null, they are considered equal.

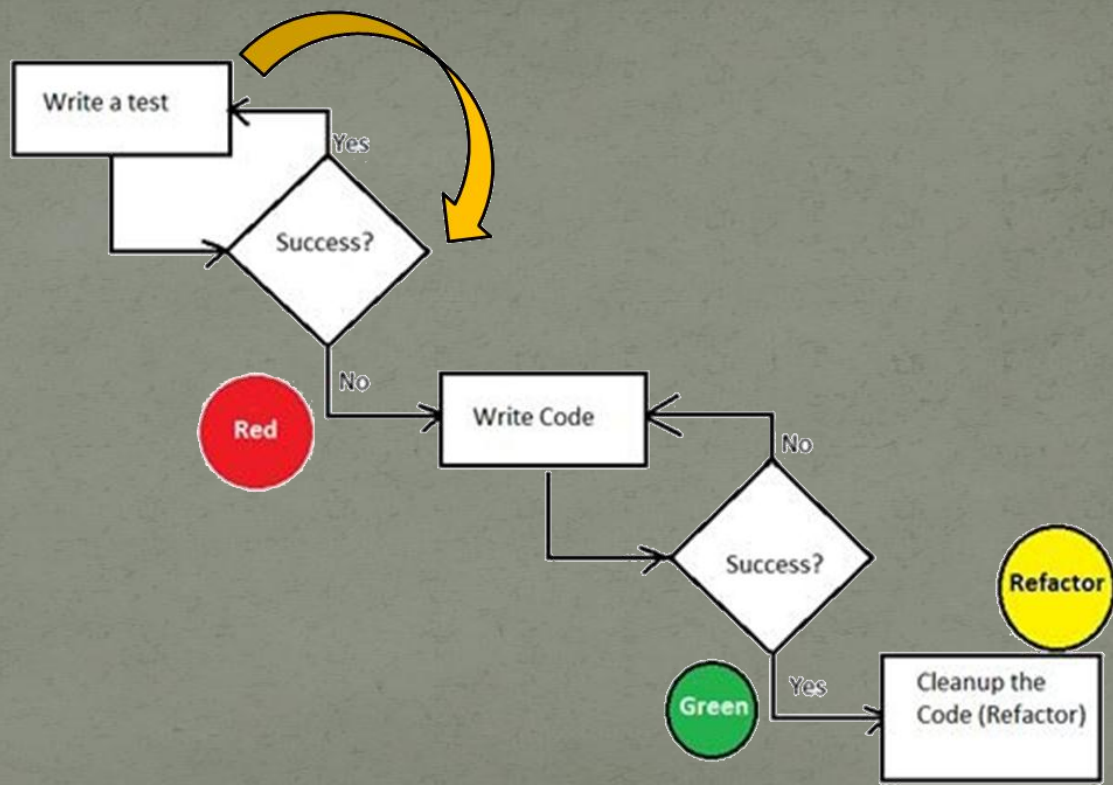


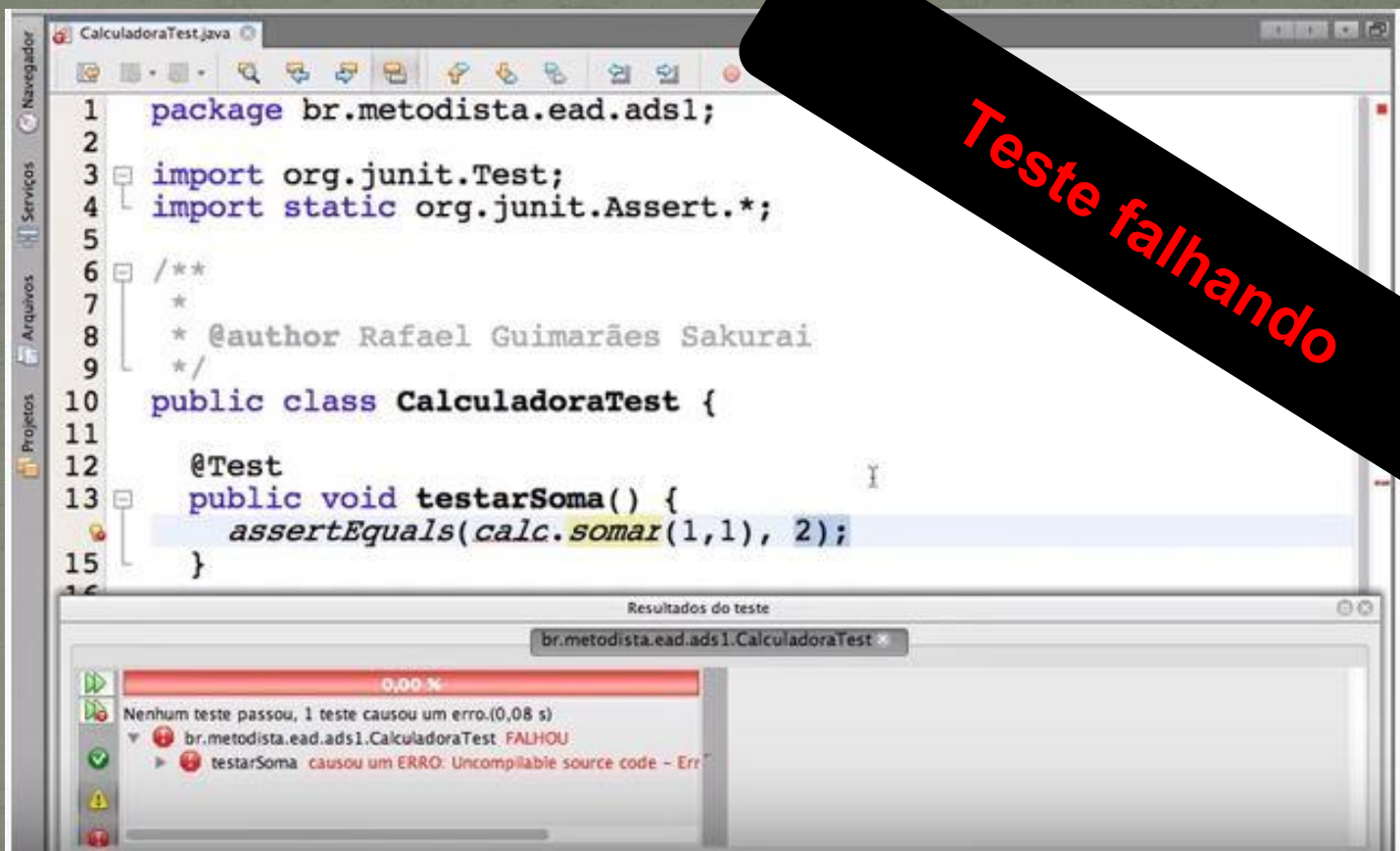
```
1 package br.metodista.ead.adsl;  
2  
3 import org.junit.Test;  
4 import static org.junit.Assert.*;  
5  
6 /**  
7  *  
8  * @author Rafael Guimarães Sakurai  
9  */  
10 public class CalculadoraTest {  
11  
12     @Test  
13     public void testarSoma() {  
14         assertEquals(2, 2);  
15     }  
16  
17 }  
18
```

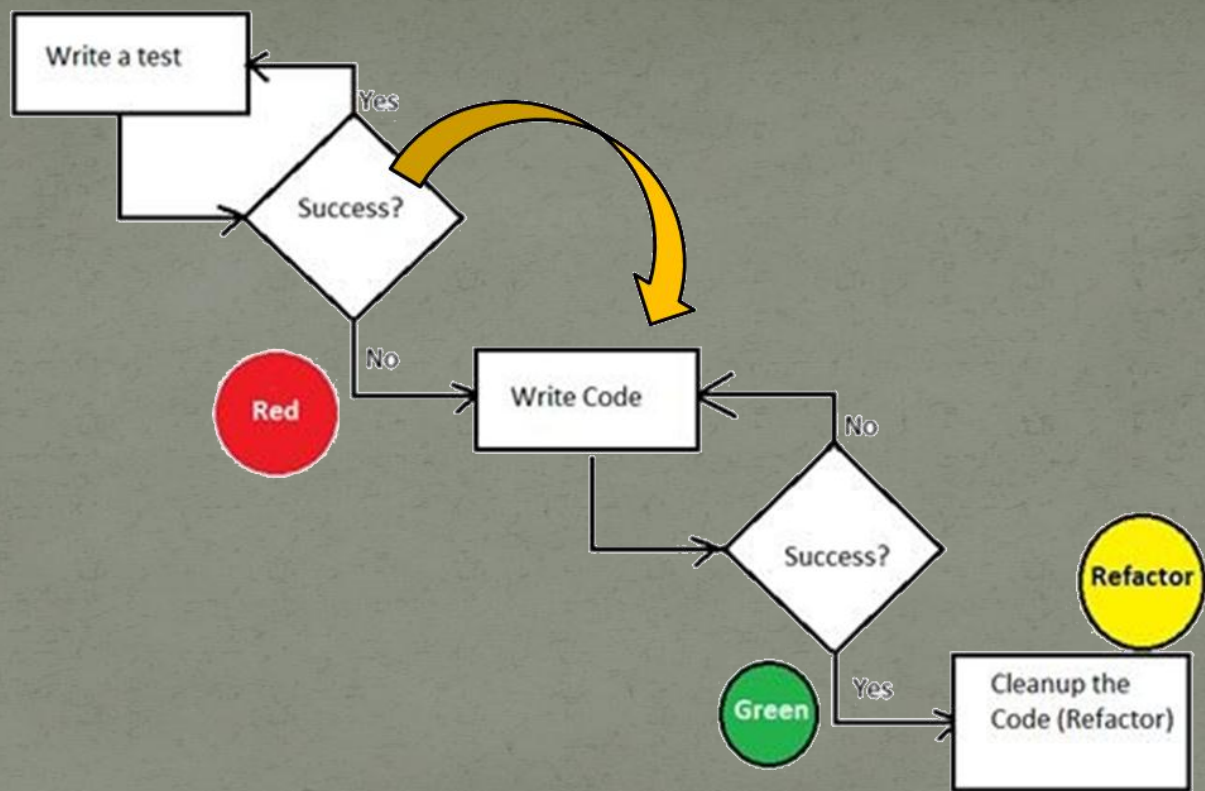


Simulação Novo Teste



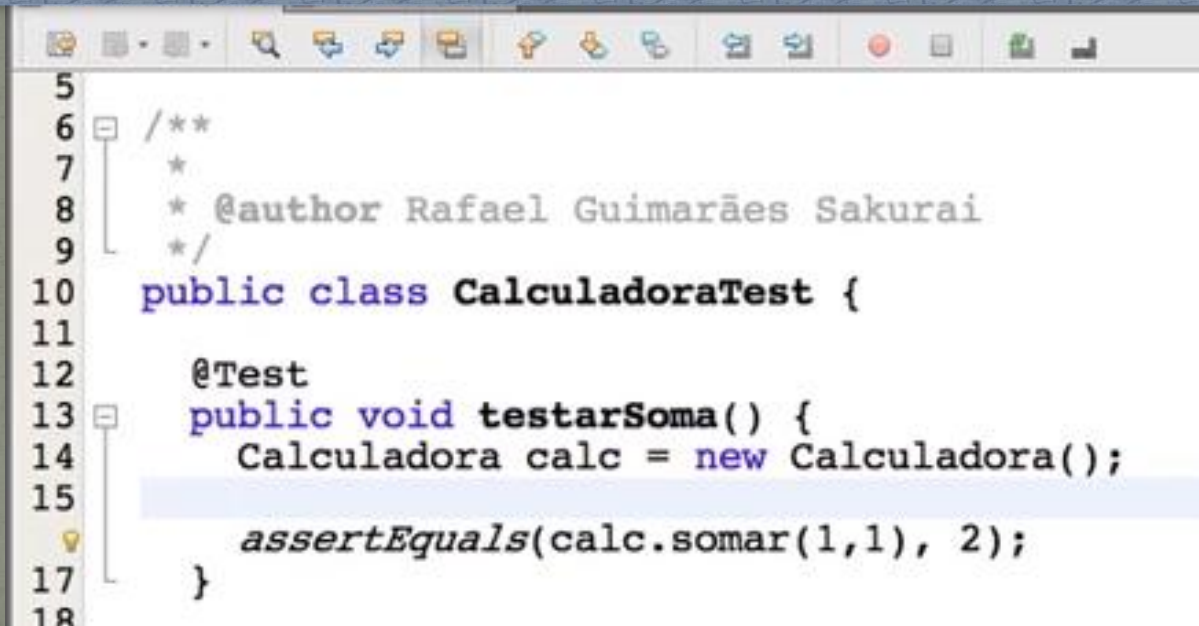






Teste passando

Calculadora calc = new Calculadora();

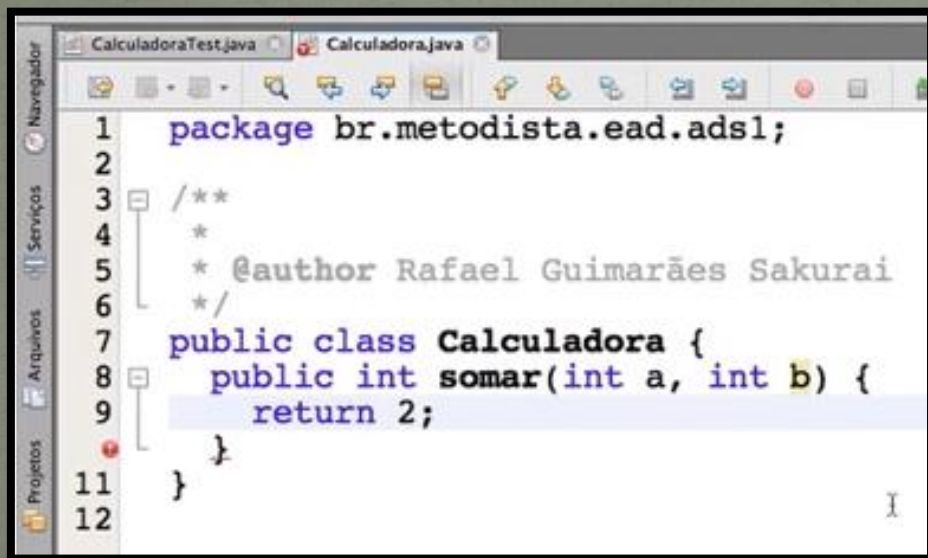


The screenshot shows a code editor with a toolbar at the top. The code is as follows:

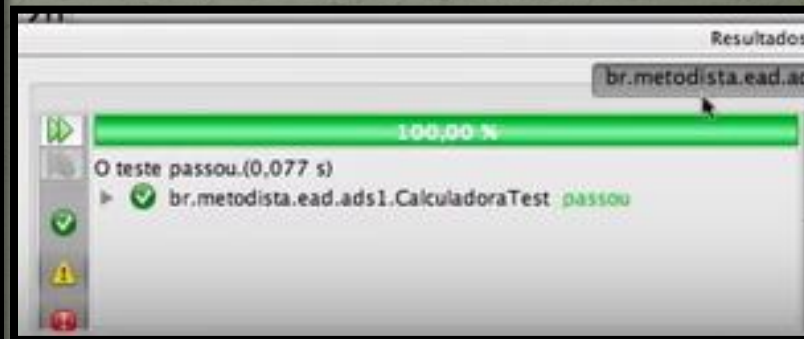
```
5
6 /**
7  *
8  * @author Rafael Guimarães Sakurai
9  */
10 public class CalculadoraTest {
11
12     @Test
13     public void testarSoma() {
14         Calculadora calc = new Calculadora();
15
16         assertEquals(calc.somar(1,1), 2);
17     }
18 }
```

Teste passando

Nova classe Calculadora



```
1 package br.metodista.ead.ads1;
2
3 /**
4  *
5  * @author Rafael Guimarães Sakurai
6  */
7 public class Calculadora {
8     public int somar(int a, int b) {
9         return 2;
10    }
11 }
12
```



Refatorar

```
5
6  /**
7   *
8   * @author Rafael Guimarães Sakurai
9   */
10 public class CalculadoraTest {
11
12     @Test
13     public void testarSoma() {
14         Calculadora calc = new Calculadora();
15
16         assertEquals(calc.somar(1,1), 2);|
17         assertEquals(calc.somar(1,0), 1);
18     }
19
20 }
```

Novo teste !



Refatorar

```
CalculadoraTest.java Calculadora.java
1 package br.metodista.ead.ads1;
2
3 /**
4  *
5  * @author Rafael Guimarães Sakurai
6  */
7 public class Calculadora {
8     public int somar(int a, int b) {
9         return 2;
10    }
11 }
12
```

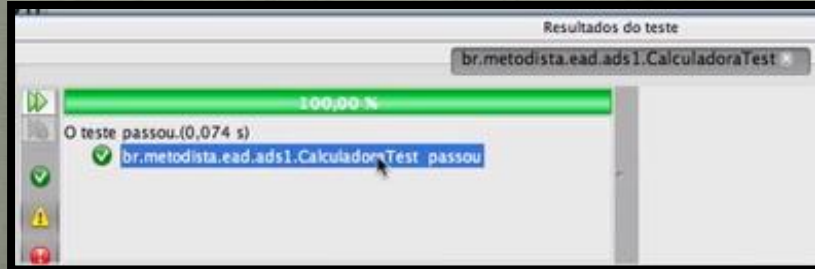
```
CalculadoraTest.java Calculadora.java
1 package br.metodista.ead.ads1;
2
3 /**
4  *
5  * @author Rafael Guimarães Sakurai
6  */
7 public class Calculadora {
8     public int somar(int a, int b) {
9         return a + b;
10    }
11 }
12
```

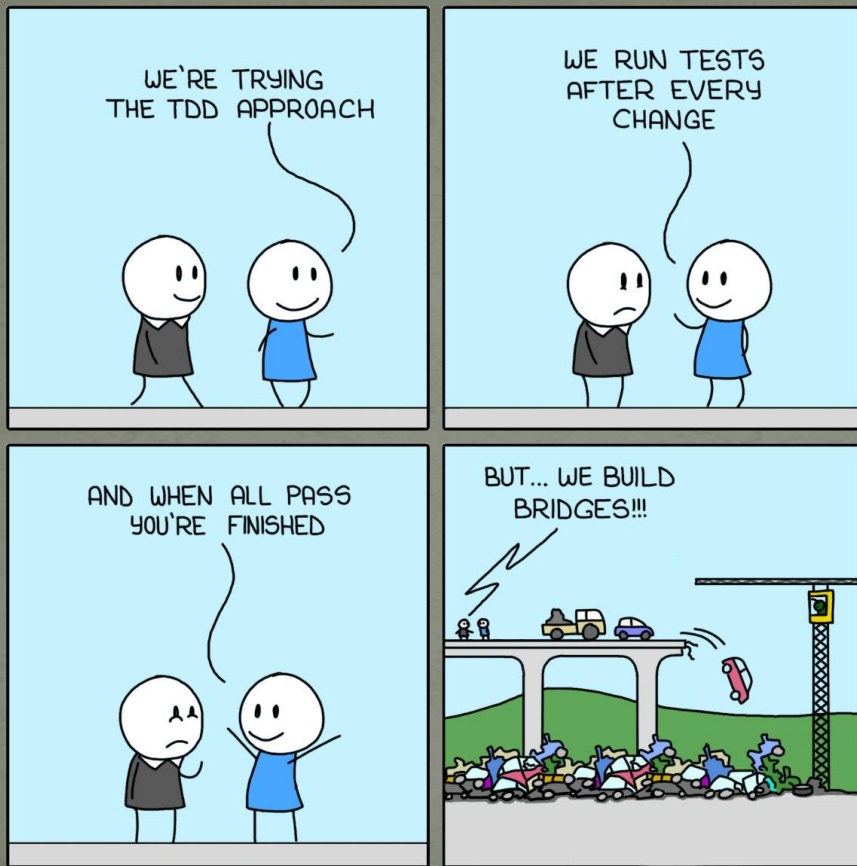
Refatorar

```
5
6  /**
7   *
8   * @author Rafael Guimarães Sakurai
9   */
10 public class CalculadoraTest {
11
12     @Test
13     public void testarSoma() {
14         Calculadora calc = new Calculadora();
15
16         assertEquals(calc.somar(1,1), 2);
17         assertEquals(calc.somar(1,0), 1);
18         assertEquals(calc.somar(1,-1), 0);
19     }
20
21 }
22
```

Refatorar

```
10 public class CalculadoraTest {  
11  
12     @Test  
13     public void testarSoma() {  
14         Calculadora calc = new Calculadora();  
15  
16         assertEquals(calc.somar(1,1), 2);  
17         assertEquals(calc.somar(1,0), 1);  
18         assertEquals(calc.somar(1,-1), 0);  
19     }  
20  
21 }  
22
```





The End.