



Teste Unitário

Junit

Herysson R. Figueiredo
herysson.figueiredo@ufn.edu.br

New Project

X



New Project

Empty Project

Generators

Maven Archetype

JavaFX

Kotlin Multiplatform

Compose Multiplatform

IDE Plugin

Android

Directory JUnit is not empty in this location

Name:

JUnit

Location:

~\IdeaProjects



Project will be created in: ~\IdeaProjects\JUnit

Create Git repository

Language:

Java

Kotlin

Groovy

HTML



Build system:

IntelliJ

Maven

Gradle

JDK:



openjdk-20 Oracle OpenJDK version 20.0.1 ▾

Add sample code

Generate code with onboarding tips

> Advanced Settings

Dependências

Localize o pom.xml e adicione a dependência ao lado.

```
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.10.0</version>
    </dependency>
</dependencies>
```

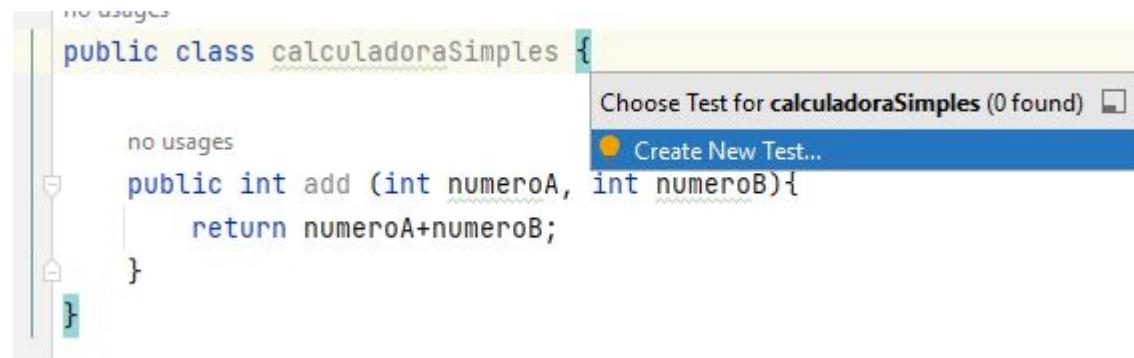
Classe

Crie uma classe que realize a soma de 2 valores inteiros e retorne o resultado

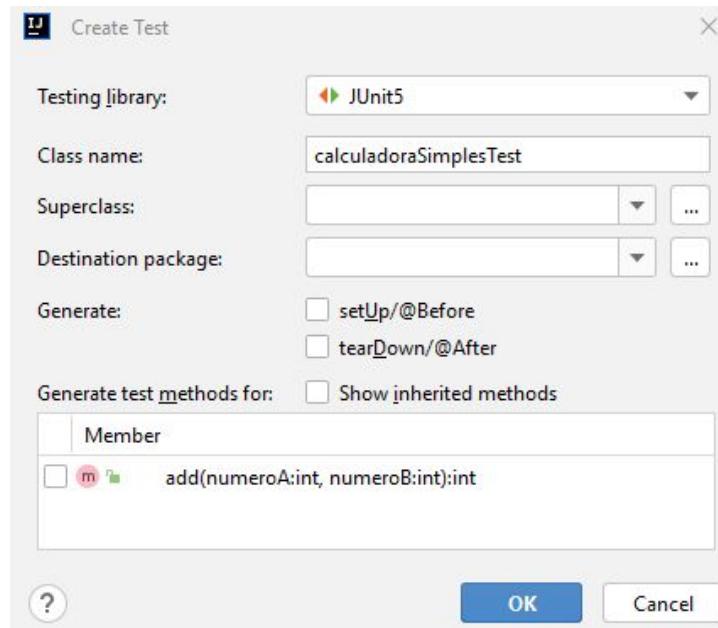
```
1 usage  
1  
public class CalculadoraSimples {  
2  
    1 usage  
3     public int add (int numeroA, int numeroB){  
4         return numeroA+numeroB;  
5     }  
6 }
```

Classe de Teste

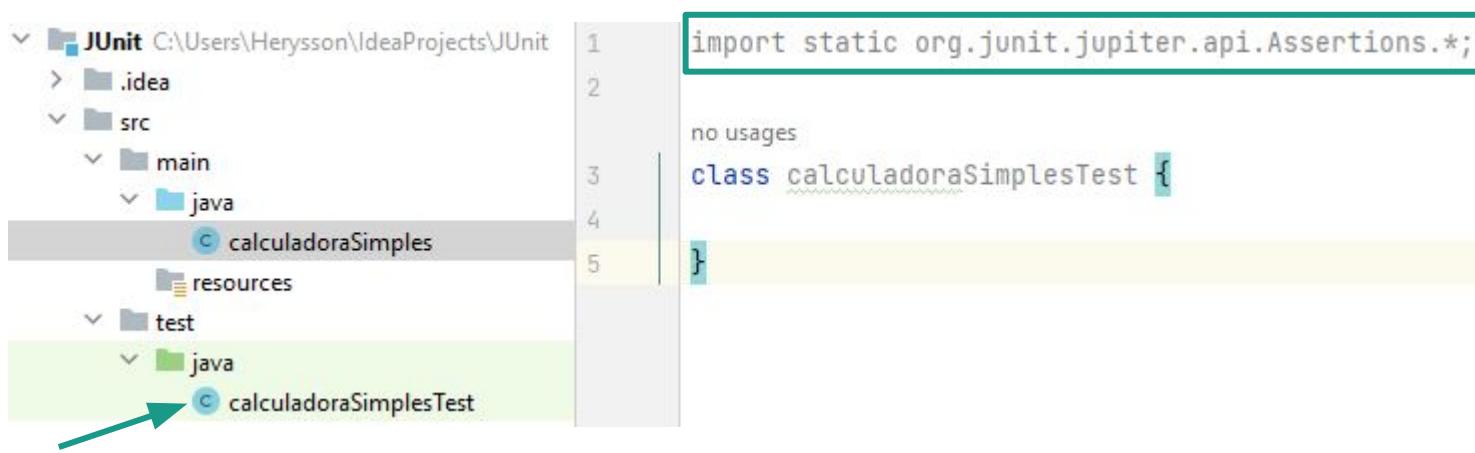
Pressione **ctrl+shift+t** para criação de um teste



Classe de Teste



Classe de Teste



The screenshot shows a file structure in an IDE:

- JUnit** (Project Root)
 - .idea
 - src
 - main
 - java
 - calculadoraSimples
 - test
 - java
 - calculadoraSimplesTest

The `calculadoraSimplesTest` class is selected in the `test/java` directory.

The code editor displays the following Java code:

```
import static org.junit.jupiter.api.Assertions.*;  
no usages  
class calculadoraSimplesTest {  
}
```

A green arrow points from the bottom left towards the `calculadoraSimplesTest` icon in the file tree.

Classe de Teste

```
1 import org.junit.jupiter.api.Test;  
2  
3 import static org.junit.jupiter.api.Assertions.*;  
4  
5 ➤ class CalculadoraSimplesTest {  
6     //cada teste deve testar somente uma unica coisa em um unico cenário  
7     //o nome do teste deve descrever o cenário e o resultado  
8     @Test  
9     void doisMaisDoisIgualQuatro(){  
10        //CalculadoraSimples calculadora = new CalculadoraSimples();  
11        var calculadora = new CalculadoraSimples();  
12        assertEquals( expected: 4,calculadora.add( numeroA: 2, numeroB: 2));  
13    }  
14 }
```



Classe de Teste

```
1 import org.junit.jupiter.api.Test;  
2  
3 import static org.junit.jupiter.api.Assertions.*;  
4  
5 ➤ class CalculadoraSimplesTest {  
6     //cada teste deve testar somente uma unica coisa em um unico cenário  
7     //o nome do teste deve descrever o cenário e o resultado  
8     @Test  
9         ➤ Run 'doisMaisDoisIgualQuatro'  
10            Ctrl+Shift+F10  
11         ➤ Debug 'doisMaisDoisIgualQuatro'  
12         ➤ Run 'doisMaisDoisIgualQuatro' with Coverage  
13             Modify Run Configuration...  
14 }  
    adora = new CalculadoraSimples();  
    adoraSimples();  
    adora.add( numeroA: 2, numeroB: 2));
```

Classe de Teste

The screenshot shows a Java code editor with a test class named `CalculadoraSimplesTest`. The code includes imports for `org.junit.jupiter.api.Test` and `org.junit.jupiter.api.Assertions.*`, and a class definition with a `@Test` annotation. A tooltip for the `@Test` annotation is displayed, stating: `//cada teste deve testar somente uma unica coisa em um unico cenário` and `//o nome do teste deve descrever o cenário e o resultado`. Below the code, a run configuration is selected in the toolbar, labeled `Run 'doisMaisDoisIgualQuatro'`. The run results show one test passed: `doisMaisDoisIgualQuatro()` took 13 ms and was executed by `C:\Users\Herysson\.jdks\openjdk-20.0.1\bin\java.exe`. The status message at the bottom right indicates `Process finished with exit code 0`.

```
1 import org.junit.jupiter.api.Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class CalculadoraSimplesTest {
6     //cada teste deve testar somente uma unica coisa em um unico cenário
7     //o nome do teste deve descrever o cenário e o resultado
8     @Test
9     Run 'doisMaisDoisIgualQ...0' Ctrl+Shift+F10
10    Calculadora = new CalculadoraSimples();
11
12    Run: CalculadoraSimplesTest.doisMaisDoisIgualQuatro
13    ✓ CalculadoraSimplesTest
14        ✓ doisMaisDoisIgualQuatro() 13 ms
15
16 C:\Users\Herysson\.jdks\openjdk-20.0.1\bin\java.exe ...
17
18 Process finished with exit code 0
```

Classe

Alterando a classe CalculadoraSimples

```
1  public class CalculadoraSimples {  
2  
3      public int add (int numeroA, int numeroB){  
4          return numeroA+numeroB;  
5      }  
6  }
```

Classe de Teste

The screenshot shows a code editor with a Java test class named `CalculadoraSimplesTest`. The code includes imports for `org.junit.jupiter.api.Test` and `org.junit.jupiter.api.Assertions.*`, and a class definition with a `@Test` annotation. A tooltip for the `@Test` annotation provides the following notes:

- //cada teste deve testar somente uma unica coisa em um unico cenário
- //o nome do teste deve descrever o cenário e o resultado

The IDE interface includes a toolbar with icons for Run, Debug, and Modify Run. A dropdown menu shows the selected run configuration: `Run 'doisMaisDoisIgualQuatro'`. The status bar at the bottom right shows the command `C:\Users\Herysson\.jdks\openjdk-20.0.1\bin\java.exe ...`.

Output Window Content:

```
Tests failed: 1 of 1 test - 25 ms
C:\Users\Herysson\.jdks\openjdk-20.0.1\bin\java.exe ...
org.opentest4j.AssertionFailedError:
Expected :4
Actual   :0
<Click to see differences>
```

Classe

Alterando a classe CalculadoraSimples

```
1  public class CalculadoraSimples {  
2  
3      2 usages  
4      public int add (int numeroA, int numeroB){  
5          return numeroA*numeroB;  
6      }  
7  }
```

Classe de Teste

```
1 import org.junit.jupiter.api.Test;  
2  
3 import static org.junit.jupiter.api.Assertions.*;  
4  
5 class CalculadoraSimplesTest {  
6     //cada teste deve testar somente uma unica coisa em um unico cenário  
7     //o nome do teste deve descrever o cenário e o resultado  
8     @Test  
9     void doisMaisDoisIgualQuatro(){  
10         var calculadora = new CalculadoraSimples();  
11         assertTrue( condition: calculadora.add( numeroA: 2, numeroB: 2)==4);  
12     }  
13     @Test  
14     void tresMaisSeteIgualDez(){  
15         var calculadora = new CalculadoraSimples();  
16         assertEquals( expected: 10,calculadora.add( numeroA: 3, numeroB: 7));  
17     }  
18 }  
19 }
```



Classe de Teste

A screenshot of a Java code editor showing a context menu for a test method. The menu items are:

- Run 'CalculadoraSimplesTest' Ctrl+Shift+F10
- Debug 'CalculadoraSimplesTest'
- Run 'CalculadoraSimplesTest' with Coverage
- Modify Run Configuration...

The code editor shows a portion of a test class:

```
1 import org.junit.jupiter.api.Test;  
2  
3 import static org.junit.jupiter.api.Assertions.*;  
4  
5 @Test  
6 void somaDoisNumeros() {  
7     var calculadora = new CalculadoraSimples();  
8     assertTrue( condition: calculadora.add( numeroA: 2, numeroB: 2)==4 );  
9 }  
10  
11 @Test  
12 void tresMaisSeteIgualDez(){  
13     var calculadora = new CalculadoraSimples();  
14     assertEquals( expected: 10,calculadora.add( numeroA: 3, numeroB: 7 ));  
15 }  
16  
17 }
```

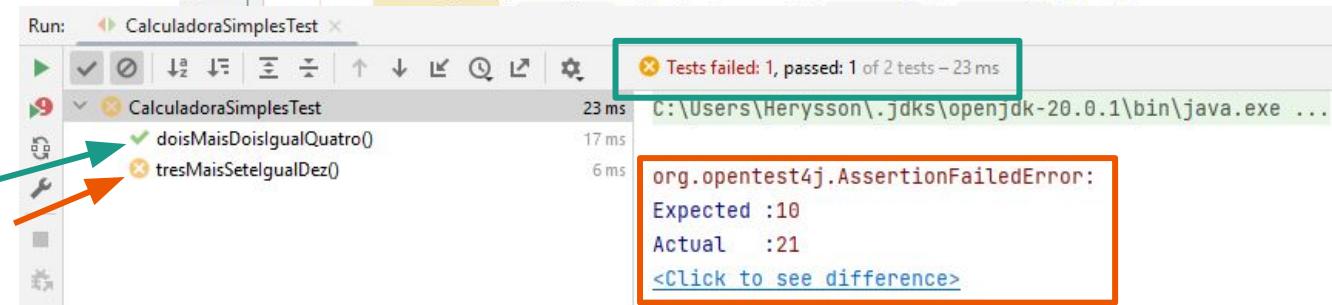
A tooltip is visible on the right side of the screen, containing the text: "coisa em um unico cenário e o resultado".

Classe de Teste

```
1 import org.junit.jupiter.api.Test;  
2  
3 import static org.junit.jupiter.api.Assertions.*;  
4  
5 class CalculadoraSimplesTest {  
6     //cada teste deve testar somente uma unica coisa em um unico cenário  
7     //o nome do teste deve descrever o cenário e o resultado  
8     @Test  
9     void doisMaisDoisIgualQuatro(){  
10        var calculadora = new CalculadoraSimples();  
11        assertTrue( condition: calculadora.add( numeroA: 2, numeroB: 2)==4);  
12    }  
13  
14    @Test  
15    void tresMaisSeteIgualDez(){  
16        var calculadora = new CalculadoraSimples();  
17        assertEquals( expected: 10,calculadora.add( numeroA: 3, numeroB: 7));  
18    }  
19}
```

Classe de Teste

```
1 import org.junit.jupiter.api.Test;  
2  
3 import static org.junit.jupiter.api.Assertions.*;  
4  
5 class CalculadoraSimplesTest {  
6     //cada teste deve testar somente uma unica coisa em um unico cenário  
7     //o nome do teste deve descrever o cenário e o resultado  
8     @Test  
9     void doisMaisDoisIgualQuatro(){  
10        var calculadora = new CalculadoraSimples();
```



Classe

Alterando a classe CalculadoraSimples

```
2 usages
1 public class CalculadoraSimples {
2
3     2 usages
4         public int add (int numeroA, int numeroB){
5             return numeroA+numeroB;
6         }
7     }
```

Classe de Teste

```
1 import org.junit.jupiter.api.Test;  
2  
3 import static org.junit.jupiter.api.Assertions.*;  
4  
5 class CalculadoraSimplesTest {  
6     //cada teste deve testar somente uma unica coisa em um unico cenário  
7     //o nome do teste deve descrever o cenário e o resultado  
8     @Test  
9     void doisMaisDoisIgualQuatro(){  
10        var calculadora = new CalculadoraSimples();  
11        assertTrue( condition: calculadora.add( numeroA: 2, numeroB: 2)==4);  
12    }  
13  
14    @Test  
15    void tresMaisSeteIgualDez(){  
16        var calculadora = new CalculadoraSimples();  
17        assertEquals( expected: 10,calculadora.add( numeroA: 3, numeroB: 7));  
18    }  
19}
```

Classe de Teste

The screenshot shows an IDE interface with a code editor and a run-time console.

Code Editor Content:

```
1 import org.junit.jupiter.api.Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class CalculadoraSimplesTest {
6     //cada teste deve testar somente uma unica coisa em um unico cenário
7     //o nome do teste deve descrever o cenário e o resultado
8     @Test
9     void doisMaisDoisIgualQuatro(){
10         var calculadora = new CalculadoraSimples();
11         assertEquals(4, calculadora.add(2, 2));
12     }
13 }
14
15
16
17
18 }
19 }
```

Run Console Output:

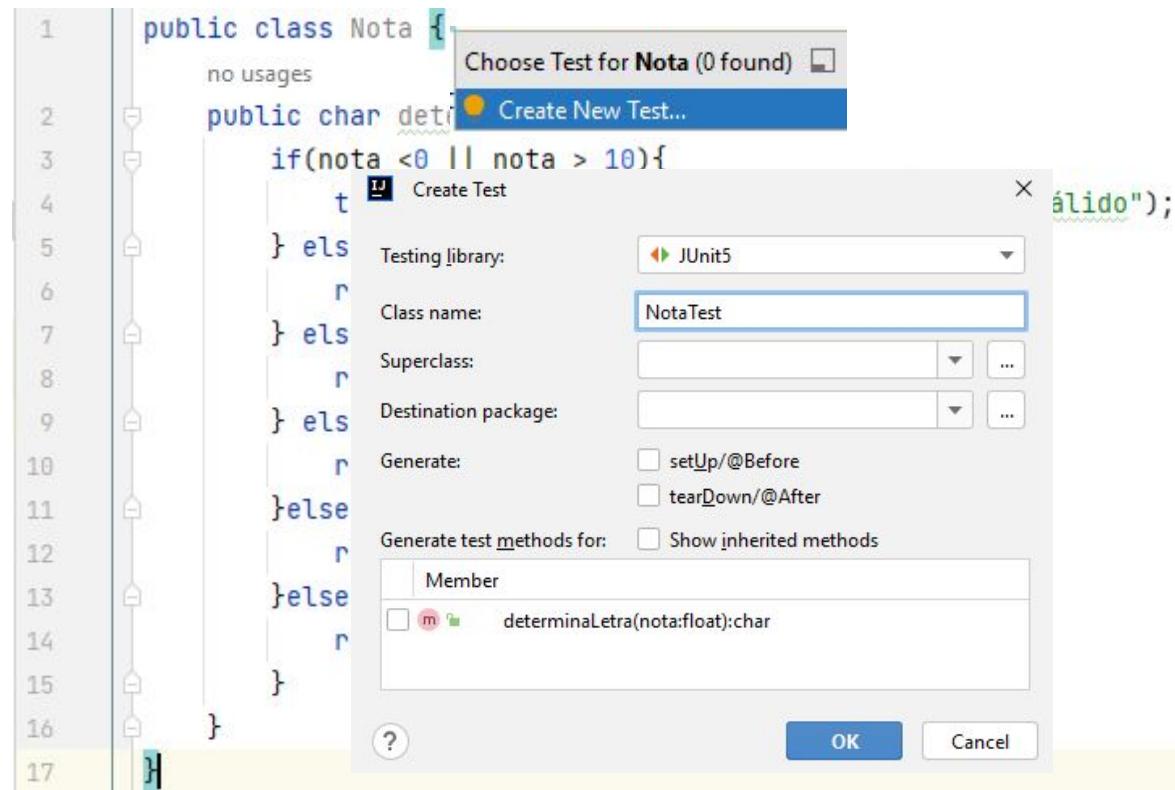
- Run: CalculadoraSimplesTest
- Tests passed: 2 of 2 tests – 16 ms
- CalculadoraSimplesTest
- doisMaisDoisIgualQuatro() 16 ms
- tresMaisSeteIgualDez() 15 ms
- C:\Users\Herysson\.jdks\openjdk-20.0.1\bin\java.exe ...
- Process finished with exit code 0

Classe : Nota

```
1  public class Nota {  
2      no usages  
3      □  
4      □  
5      □  
6      □  
7      □  
8      □  
9      □  
10     □  
11     □  
12     □  
13     □  
14     □  
15     □  
16     □  
17 }  
  
public char determinaLetra(float nota){  
    if(nota <0 || nota > 10){  
        throw new IllegalArgumentException("Valor inválido");  
    } else if (nota < 60){  
        return 'F';  
    } else if (nota < 70) {  
        return 'D';  
    } else if (nota <80) {  
        return 'C';  
    }else if (nota<90){  
        return 'B';  
    }else {  
        return 'A';  
    }  
}
```

Classe : Nota

Ctrl+Shift+t



Classe de Teste

The screenshot shows a Java code editor and a run-time interface. The code editor displays a test class named `NotaTest` with one test method, `cinquentaENoveRetornaF()`. A red arrow points to the first line of the class definition, which contains the `@Test` annotation. The run-time interface at the bottom shows the test results: one test failed, taking 17ms. The failed test is `cinquentaENoveRetornaF()`, which threw a `java.lang.IllegalArgumentException: Valor inválido`. The stack trace indicates the error occurred at line 4 of the `Nota.java` file.

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
class NotaTest {
    @Test
    void cinquentaENoveRetornaF() {
        // Test code here
    }
}
```

Run: NotaTest ×

Tests failed: 1 of 1 test – 17 ms

Notas	Teste	Tempo
!	NotaTest	17 ms
!	cinquentaENoveRetornaF()	17 ms

java.lang.IllegalArgumentException: Valor inválido

at Nota.determinaLetra(Nota.java:4)
at NotaTest.cinquentaENoveRetornaF(NotaTest.java:9) <1 internal line>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

Classe : Nota

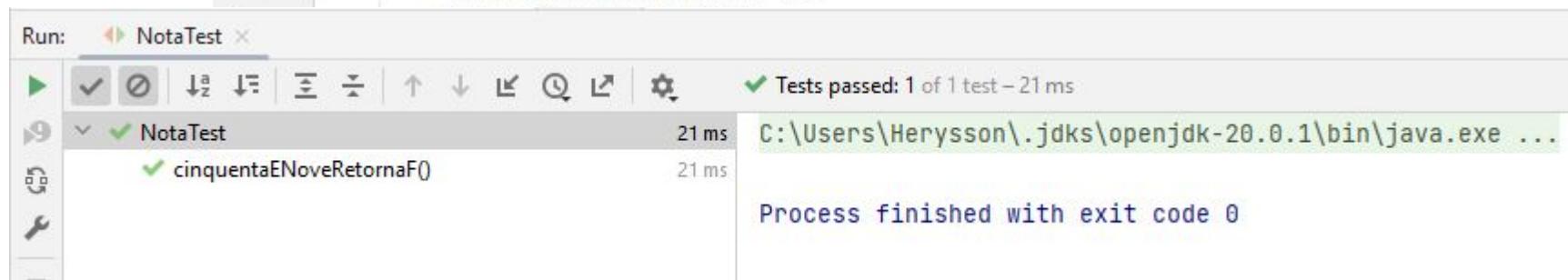
```
1  public class Nota {  
2      no usages  
3      □ public char determinaLetra(float nota){  
4          if(nota <0 || nota > 10){  
5              throw new IllegalArgumentException("Valor inválido");  
6          } else if (nota < 60){  
7              return 'F';  
8          } else if (nota < 70) {  
9              return 'D';  
10         } else if (nota <80) {  
11             return 'C';  
12         }else if (nota<90){  
13             return 'B';  
14         }else {  
15             return 'A';  
16         }  
17     }
```

Classe : Nota

```
1  public class Nota {  
2      1 usage  
3      □  □  
4      □  □  
5      □  □  
6      □  □  
7      □  □  
8      □  □  
9      □  □  
10     □  □  
11     □  □  
12     □  □  
13     □  □  
14     □  □  
15     □  □  
16     □  □  
17 } }
```

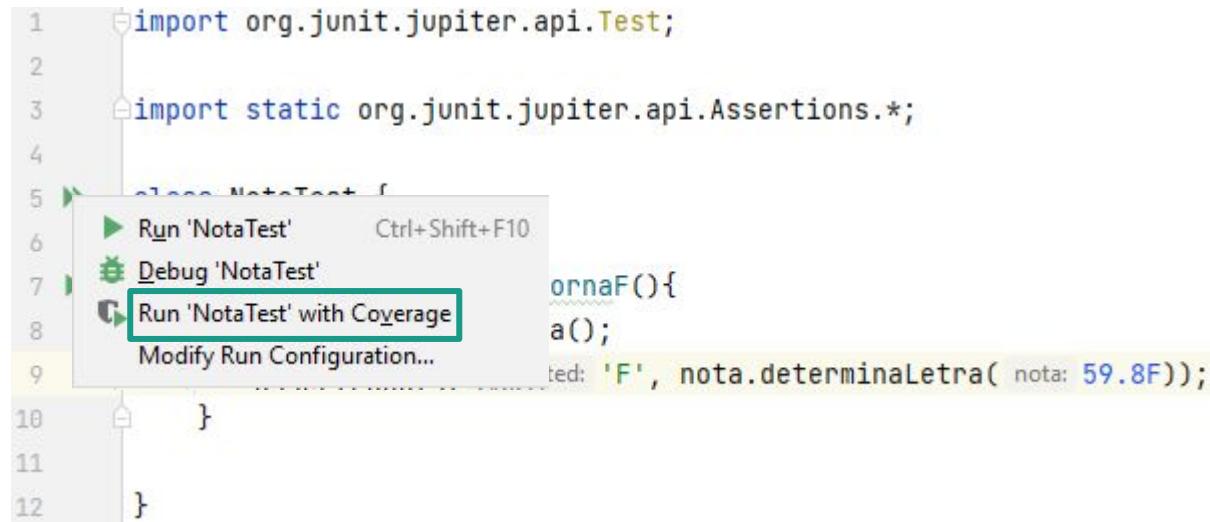
Classe de Teste

```
1 import org.junit.jupiter.api.Test;  
2  
3 import static org.junit.jupiter.api.Assertions.*;  
4  
5 ➤ class NotaTest {  
6     @Test  
7     ➤ void cinquentaENoveRetornaF(){
```



Classe de Teste

```
1 import org.junit.jupiter.api.Test;  
2  
3 import static org.junit.jupiter.api.Assertions.*;  
4  
5 class NotaTest {  
6     ▶ Run 'NotaTest'      Ctrl+Shift+F10  
7     ⚙ Debug 'NotaTest'  
8     ⚡ Run 'NotaTest' with Coverage  
9     Modify Run Configuration...  
10    ornaF(){  
11        a();  
12        ed: 'F', nota.determinaLetra( nota: 59.8F));  
13    }  
14 }
```



Classe de Teste

The screenshot shows a code editor with a Java test class named `NotaTest`. The code imports `org.junit.jupiter.api.Test` and `org.junit.jupiter.api.Assertions.*`. The class contains a single method `testNota()`.

The IDE's toolbar shows the following options:

- Run 'Nota'
- Coverage: NotaTest
- Debug 'Nota'
- Run 'Nota'

A context menu is open over the code, with the "Run 'Nota'" option highlighted.

Below the code editor is a coverage report table:

Element	Class, %	Method, %	Line, %
all	50% (1/2)	50% (1/2)	30% (4/13)
CalculadoraSimples	0% (0/1)	0% (0/1)	0% (0/1)
Nota	100% (1/1)	100% (1/1)	33% (4/12)

Classe de Teste

Construa teste para
100% de cobertura.

```
public class Nota {  
    1 usage  
    public char determinaLetra(float nota){  
        2  
        if(nota <0 || nota > 100){  
            throw new IllegalArgumentException("Valor inválido");  
        } else if (nota < 60){  
            return 'F';  
        } else if (nota < 70) {  
            return 'D';  
        } else if (nota <80) {  
            return 'C';  
        }else if (nota<90){  
            return 'B';  
        }else {  
            return 'A';  
        }  
    }  
}
```

Referências

BRAGA, Pedro Henrique Cacique. Teste de Software. Pearson Education do Brasil. São Paulo. 2016. Disponível na Biblioteca Virtual. DELAMARO, Márcio;

MALDONADO, José Carlos, JINO, Mario. Introdução ao teste de software. Elsevier. Rio de Janeiro. 2007.

PRESSMAN, Roger S. Engenharia de software. 5. Ed. Rio de Janeiro: McGraw Hill, 2002.