

# Programação Orientada a Objetos – Aula 12

Prof. Dr. Eduardo Takeo Ueda  
*[eduardo.tueda@sp.senac.br](mailto:eduardo.tueda@sp.senac.br)*

# Testes Unitários

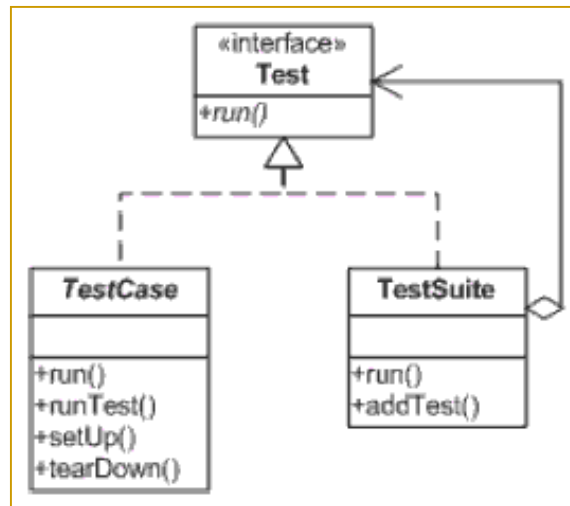
- Testar código é muito importante
  - Se seu código não funciona, ele não presta!
- Todos vocês já testam seus programas
  - Você testa uma classe quando cria algumas instâncias dela no método *main()*
- O que são testes automáticos?
  - São procedimentos que avaliam se um programa funciona como esperado, retornando respostas do tipo "**sim**" ou "**não**"

# Testes Unitários

- **Testes de unidade** são testes que testam apenas uma classe ou método, verificando se seu comportamento está de acordo com o desejado
- Em testes de unidade, verificamos a funcionalidade da classe e/ou método em questão passando o mínimo possível por outras classes ou dependências do nosso sistema
- **Unidade** é a menor parte testável de uma aplicação. Em uma linguagem de programação orientada a objetos como o Java, a menor unidade é um **método**

# Junit

- **Junit** é um framework de código aberto que nos permite escrever e executar testes para programas em Java, testes estes que podem ser facilmente repetidos
- Classes **Test**, **TestCase**, **TestSuite**, etc. oferecem a infraestrutura necessária para criar os testes



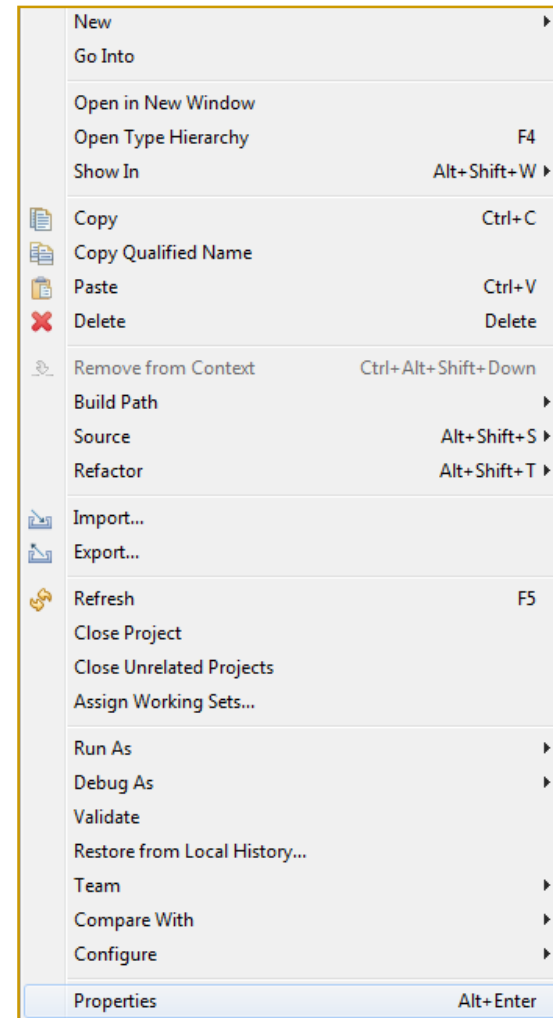
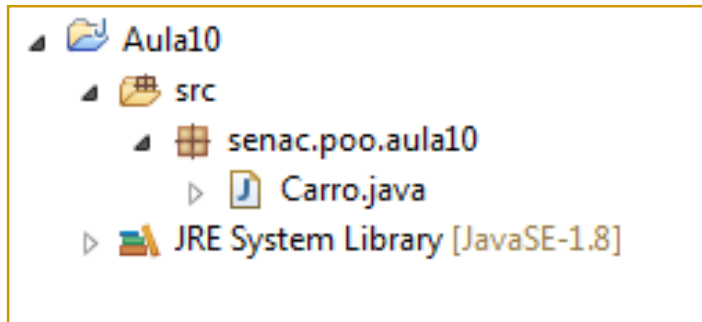
# Junit

- Métodos que usamos para testar os resultados
  - **assertTrue()**
  - **assertFalse(),**
  - **assertEquals()**
  - **assertNotEquals()**
  - **assertSame()**
  - **assertNotSame()**
  - **assertNull()**
  - **assertNotNull()**
  - **fail()**

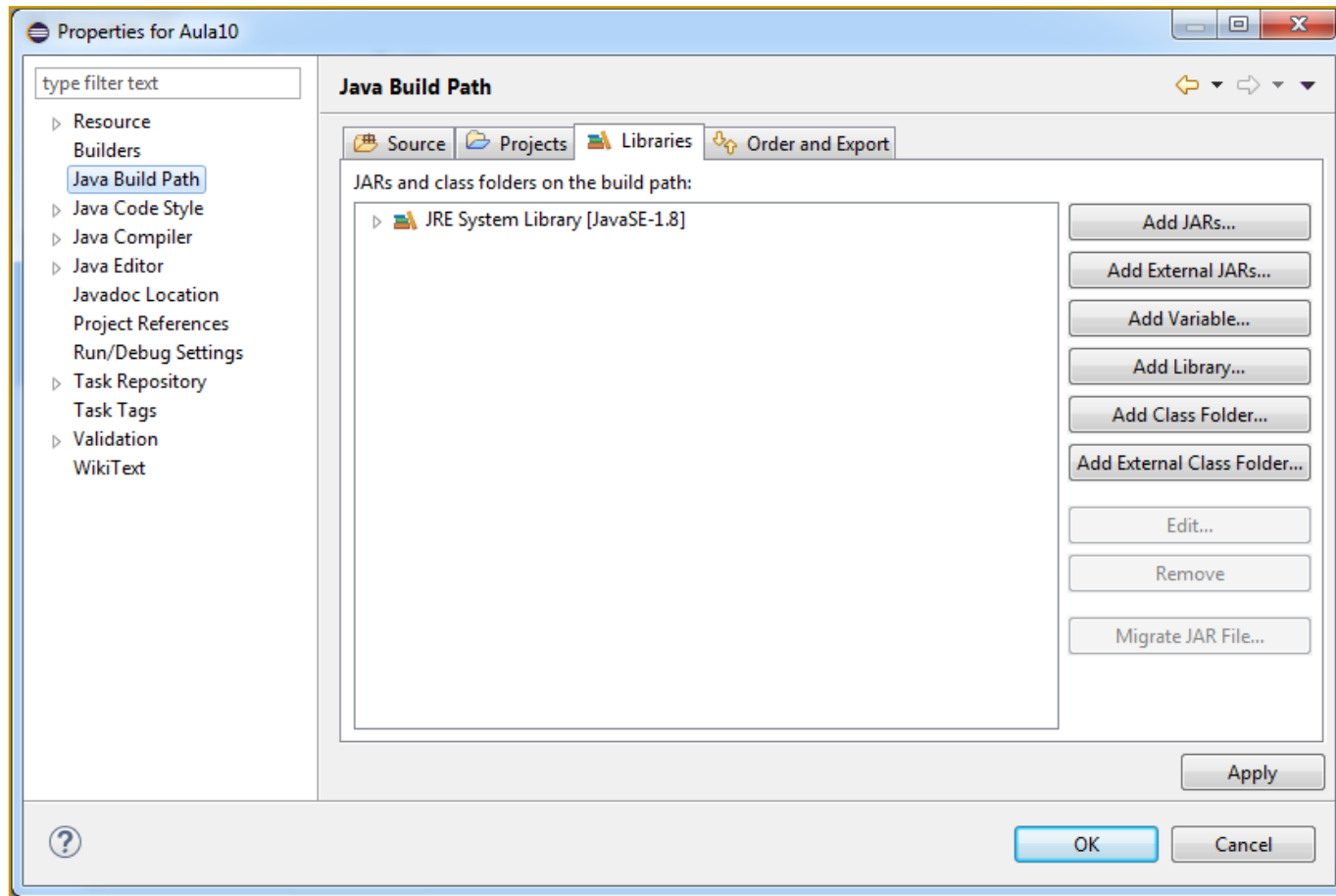
# Exemplo

```
Carro.java
1 package senac.poo.aula10;
2
3 public class Carro {
4     private int velocidade;
5     private int potencia;
6
7     public Carro(int potencia) {
8
9         this.potencia = potencia;
10        this.velocidade = 0;
11    }
12
13    public void acelerar() {
14
15        velocidade += Math.abs(potencia);
16    }
17
18    public void frear() {
19
20        velocidade /= 2;
21    }
22    public int getVelocidade() {
23
24        return velocidade;
25    }
26 }
```

# Exemplo

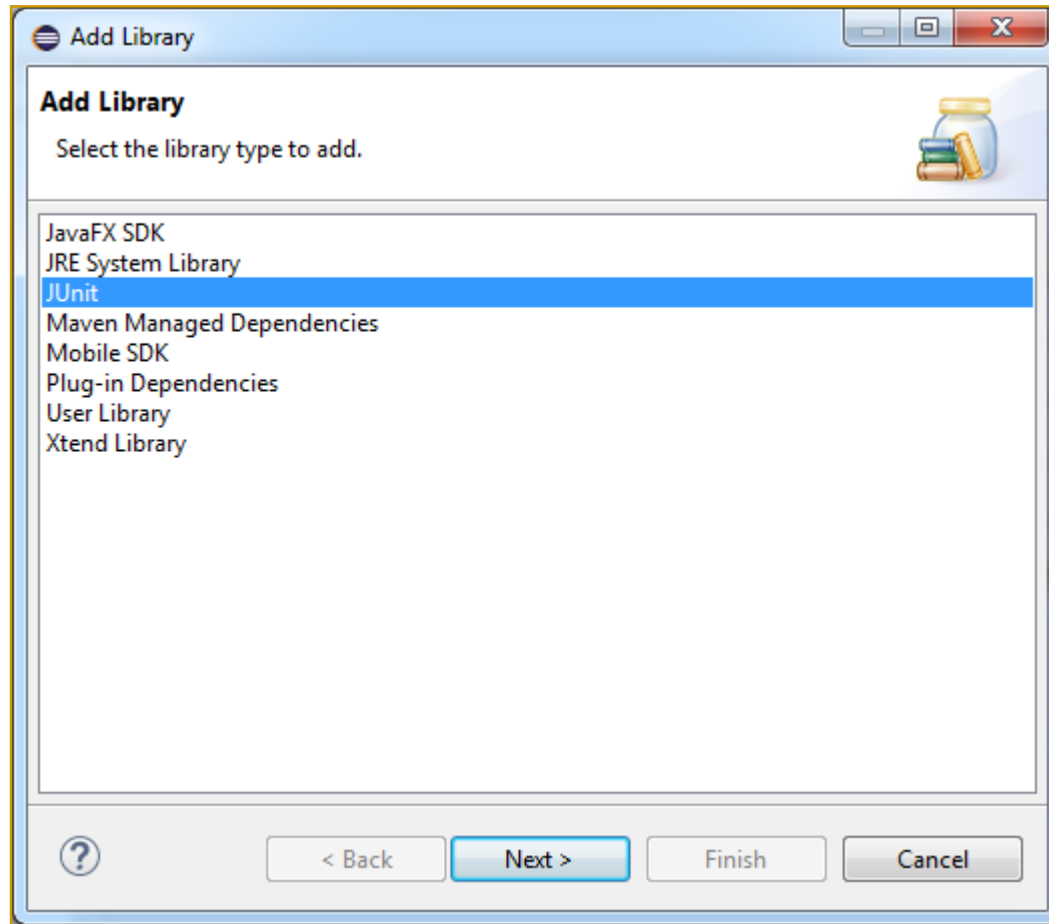


# Exemplo

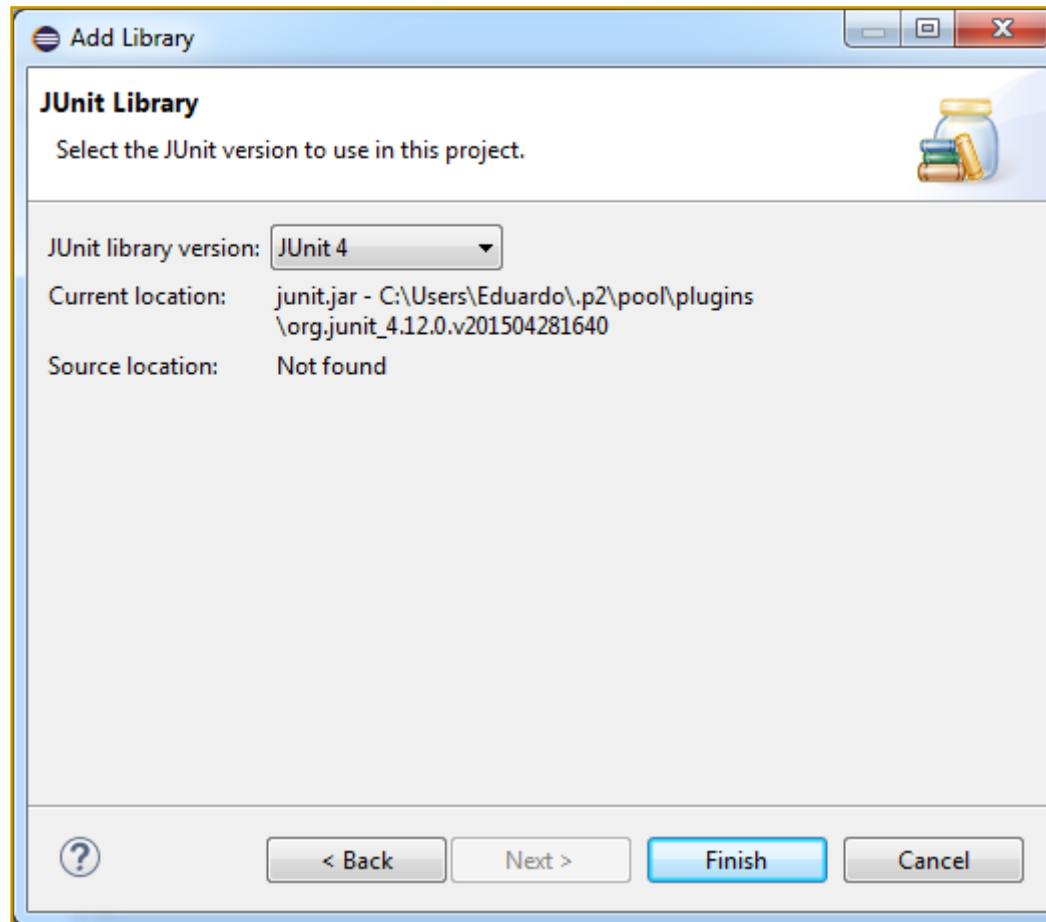




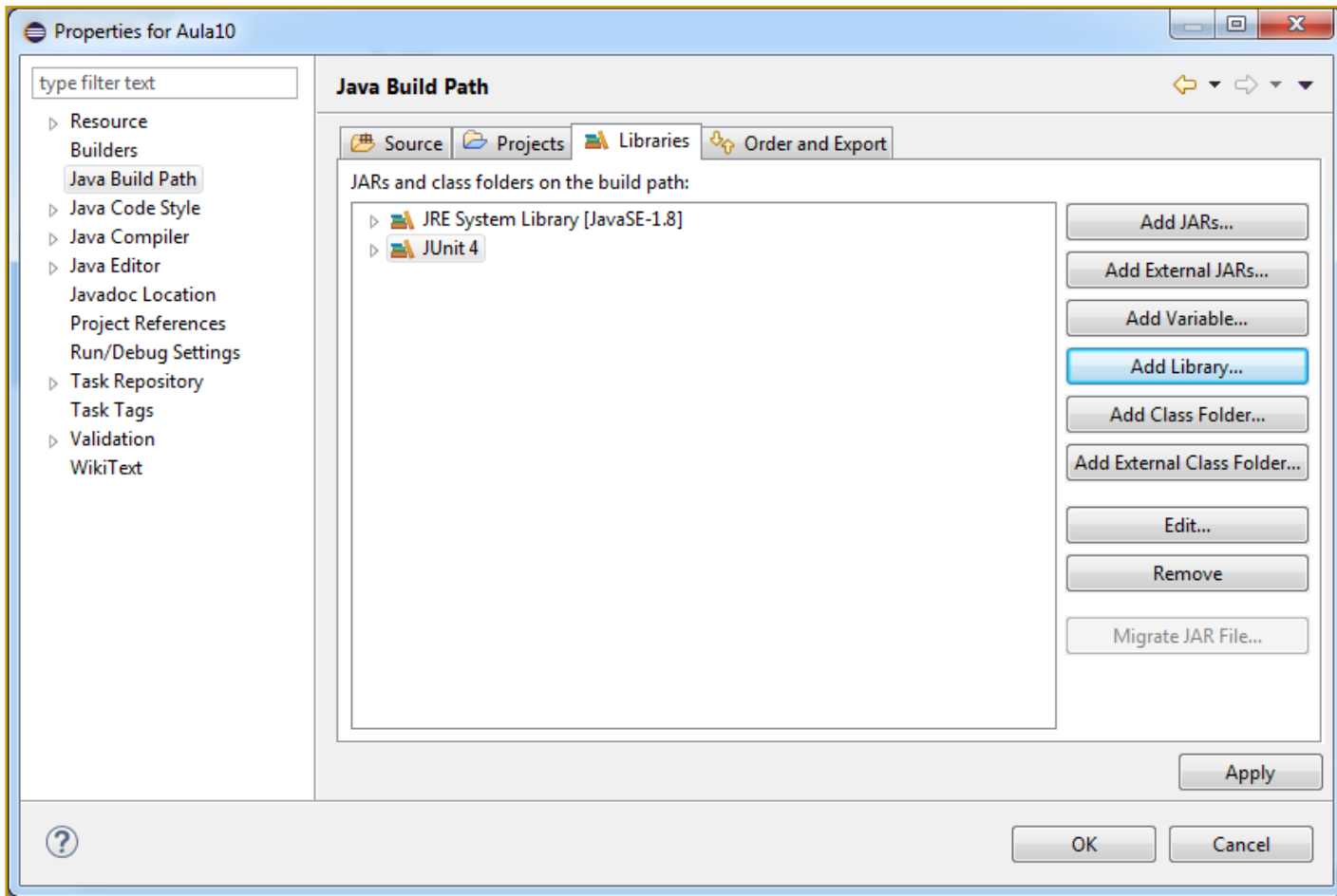
# Exemplo



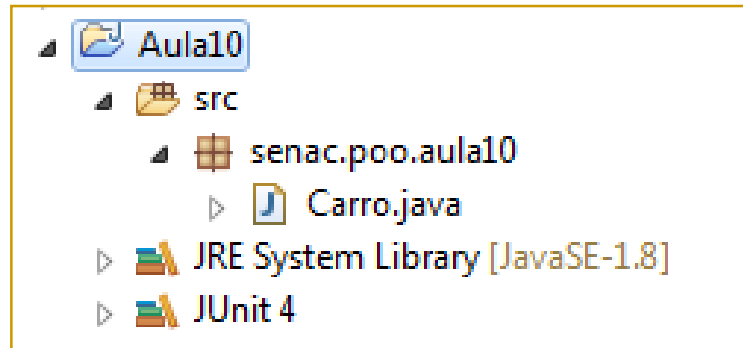
# Exemplo



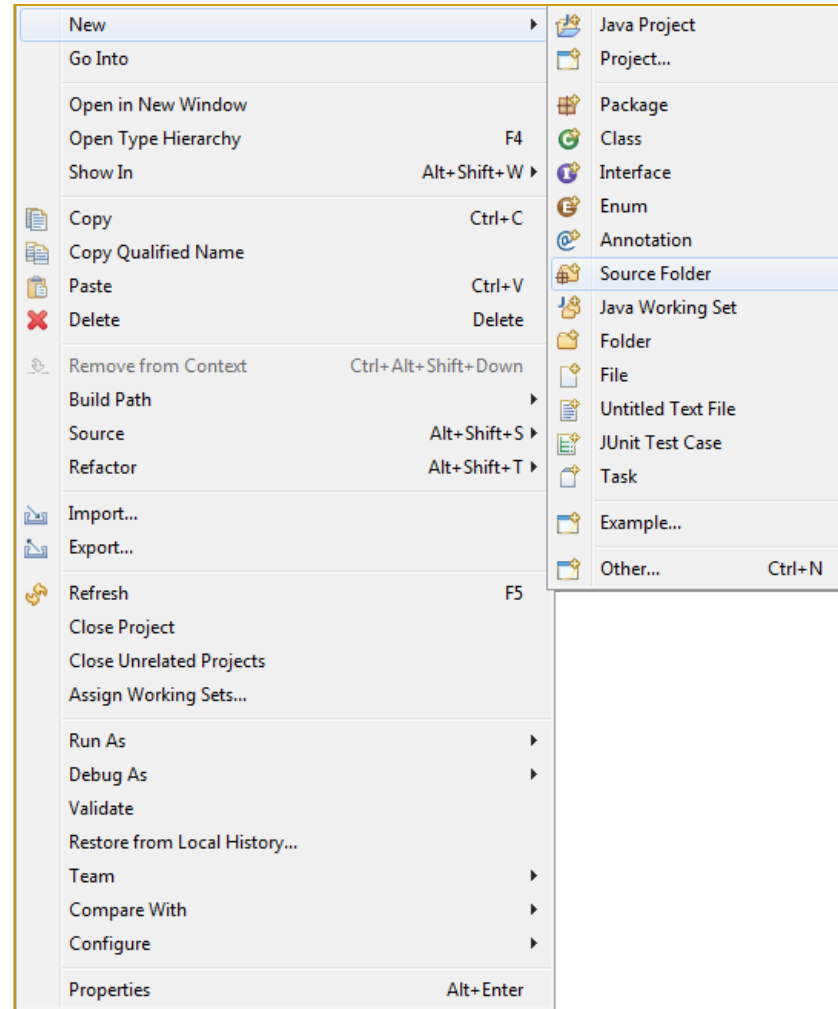
# Exemplo



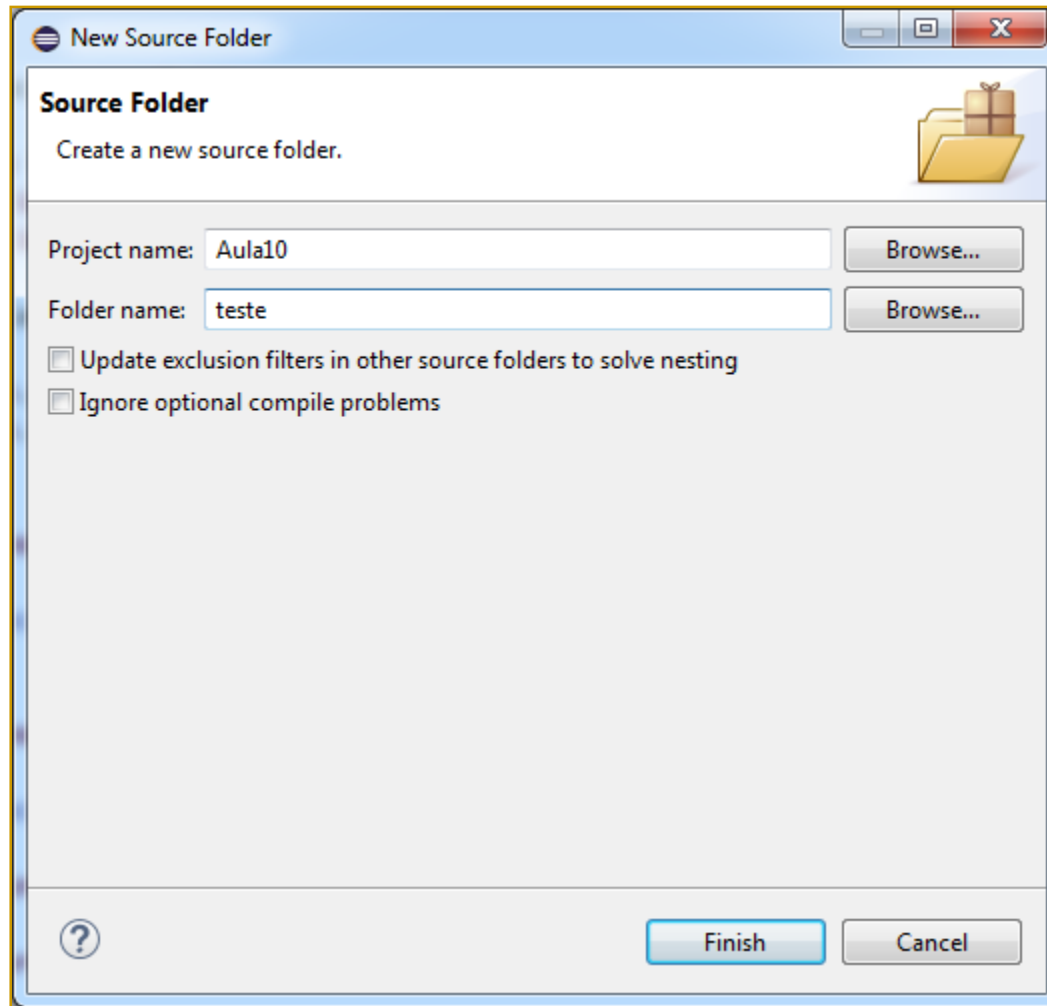
# Exemplo



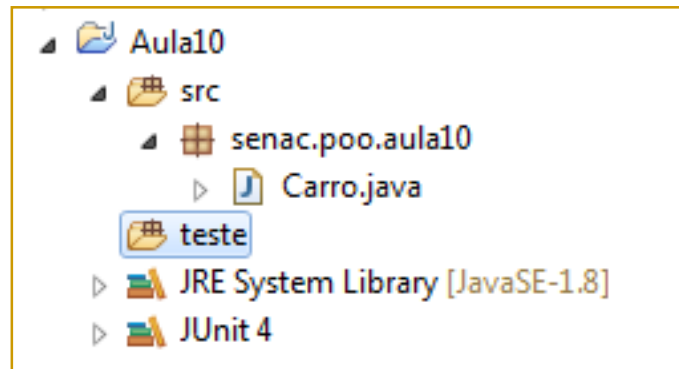
# Exemplo



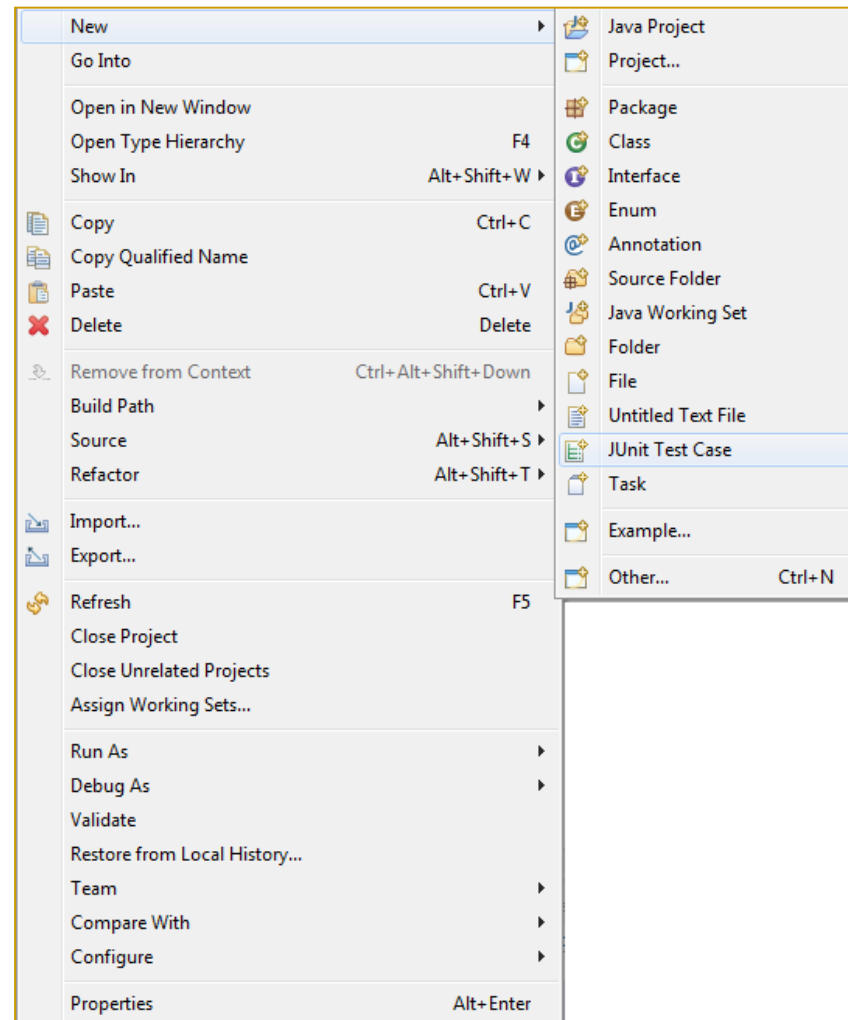
# Exemplo



# Exemplo



# Exemplo





# Exemplo

**New JUnit Test Case**

⚠ Type name is discouraged. By convention, Java type names usually start with an uppercase letter

☐ New JUnit 3 test ☒ New JUnit 4 test

Source folder:

Package:

Name:


Superclass:

Which method stubs would you like to create?

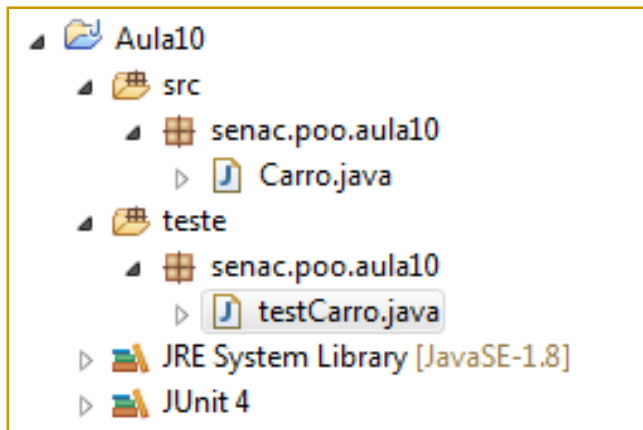
☐ setUpBeforeClass() ☐ tearDownAfterClass()  
☐ setUp() ☐ tearDown()  
☐ constructor

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

Class under test:

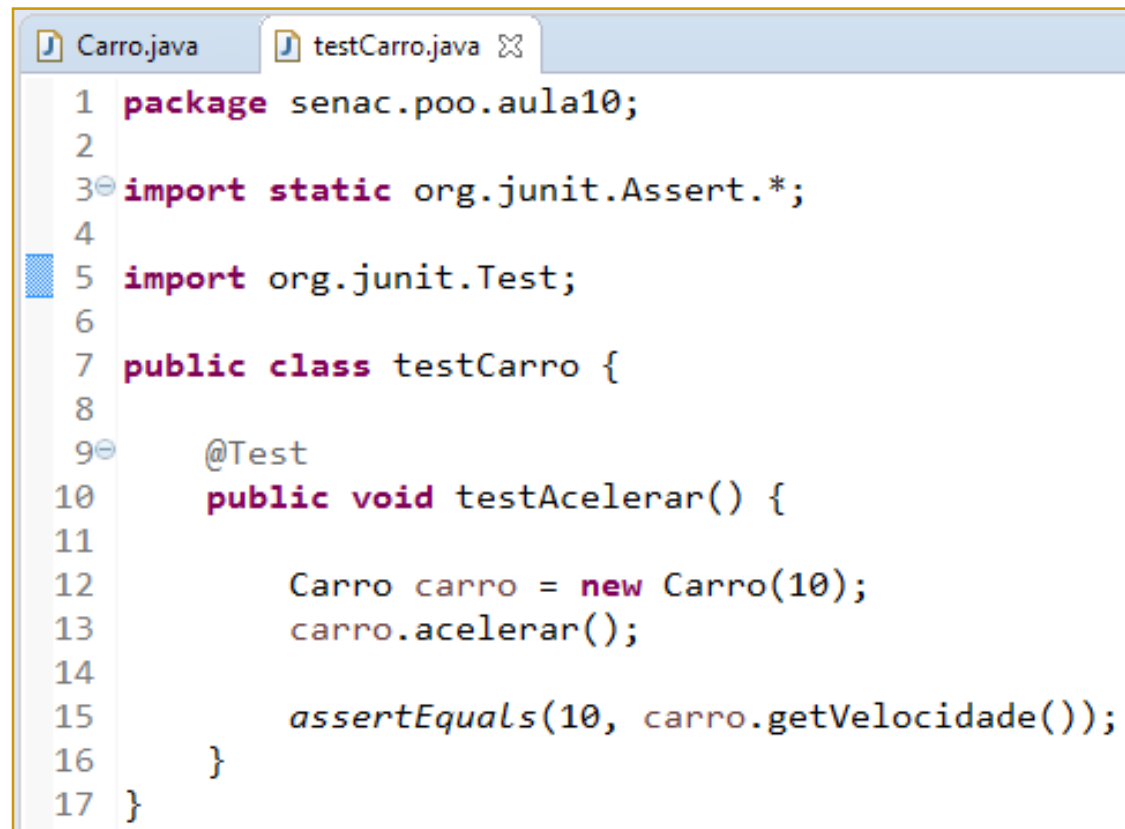


# Exemplo



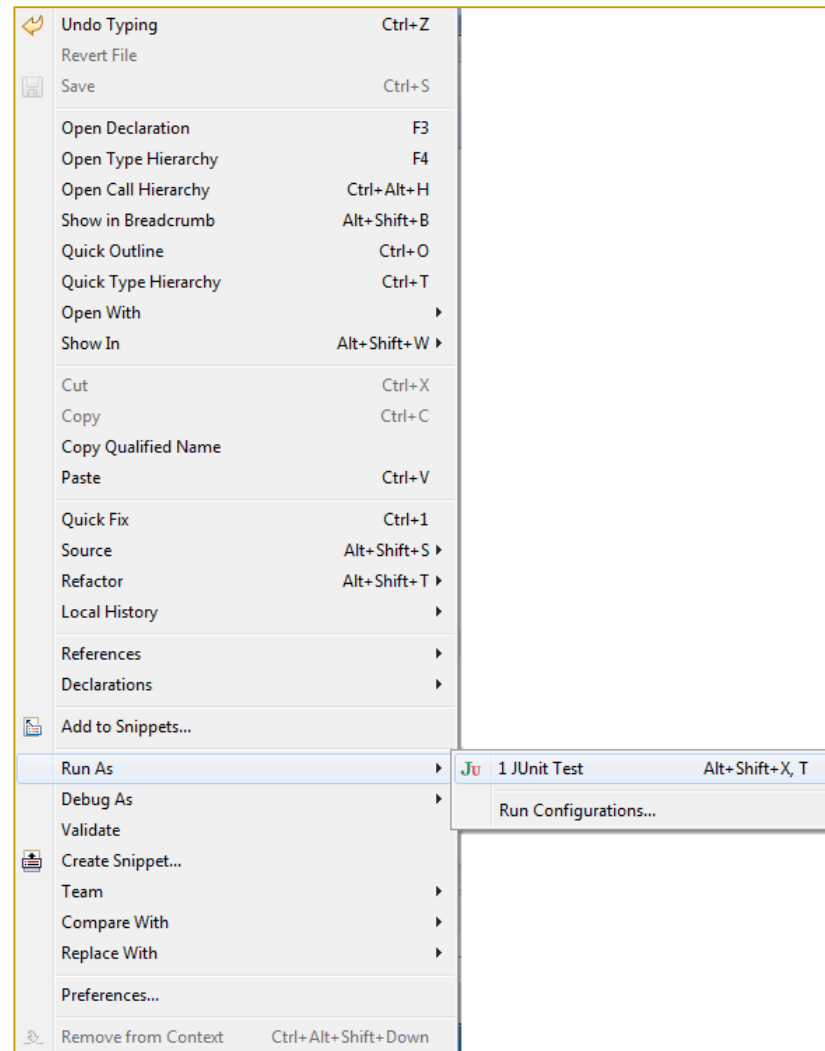
```
Carro.java  testCarro.java ✕
1  package senac.poo.aula10;
2
3+ import static org.junit.Assert.*;
6
7  public class testCarro {
8
9-     @Test
10     public void test() {
11         fail("Not yet implemented");
12     }
13
14 }
```

# Exemplo

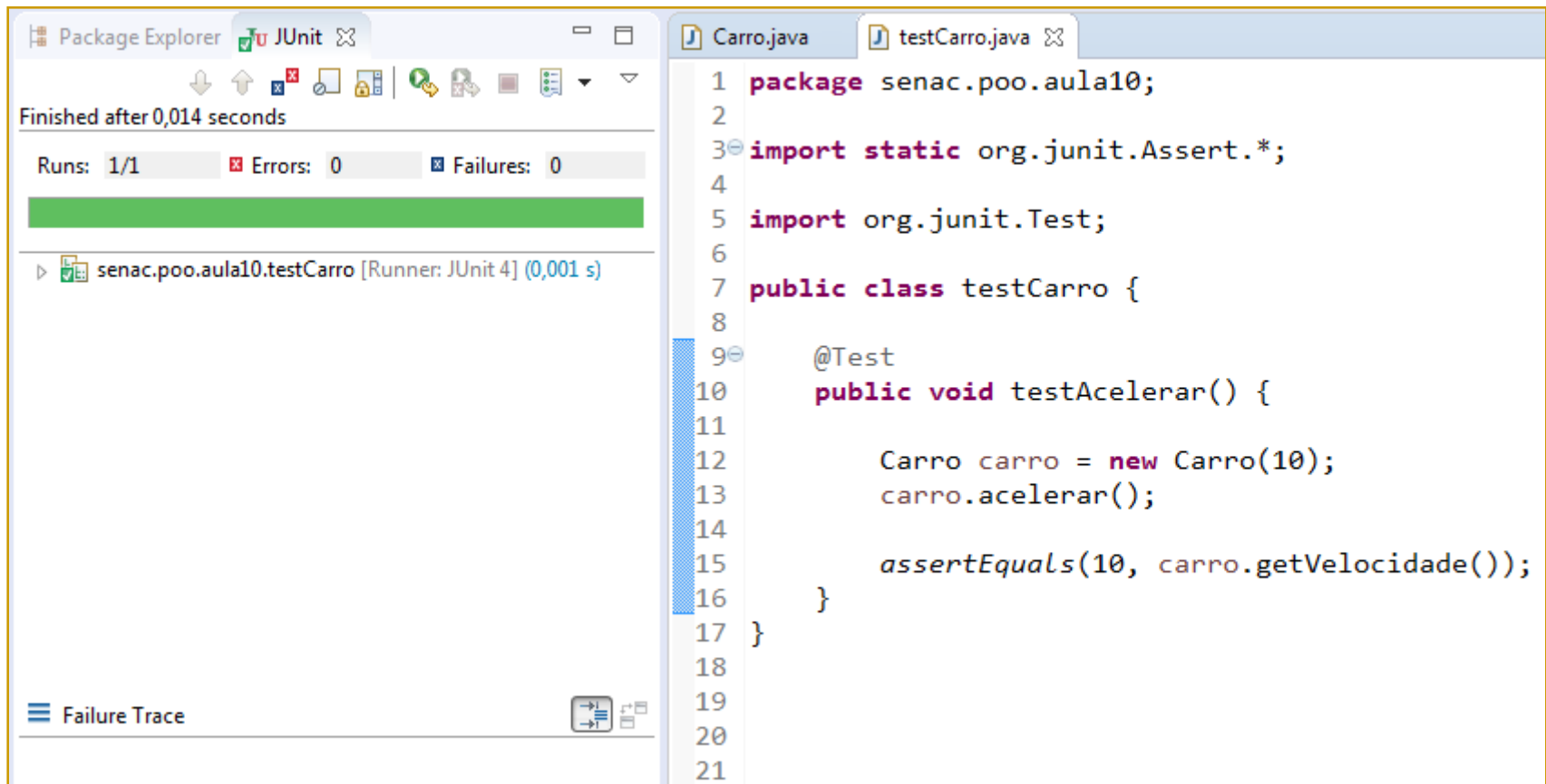


```
1 package senac.poo.aula10;
2
3 import static org.junit.Assert.*;
4
5 import org.junit.Test;
6
7 public class testCarro {
8
9     @Test
10     public void testAcelerar() {
11
12         Carro carro = new Carro(10);
13         carro.acelerar();
14
15         assertEquals(10, carro.getVelocidade());
16     }
17 }
```

# Exemplo



# Exemplo



The screenshot displays an IDE interface with two main panels. The left panel shows the JUnit test results, indicating a successful run. The right panel shows the source code for the test.

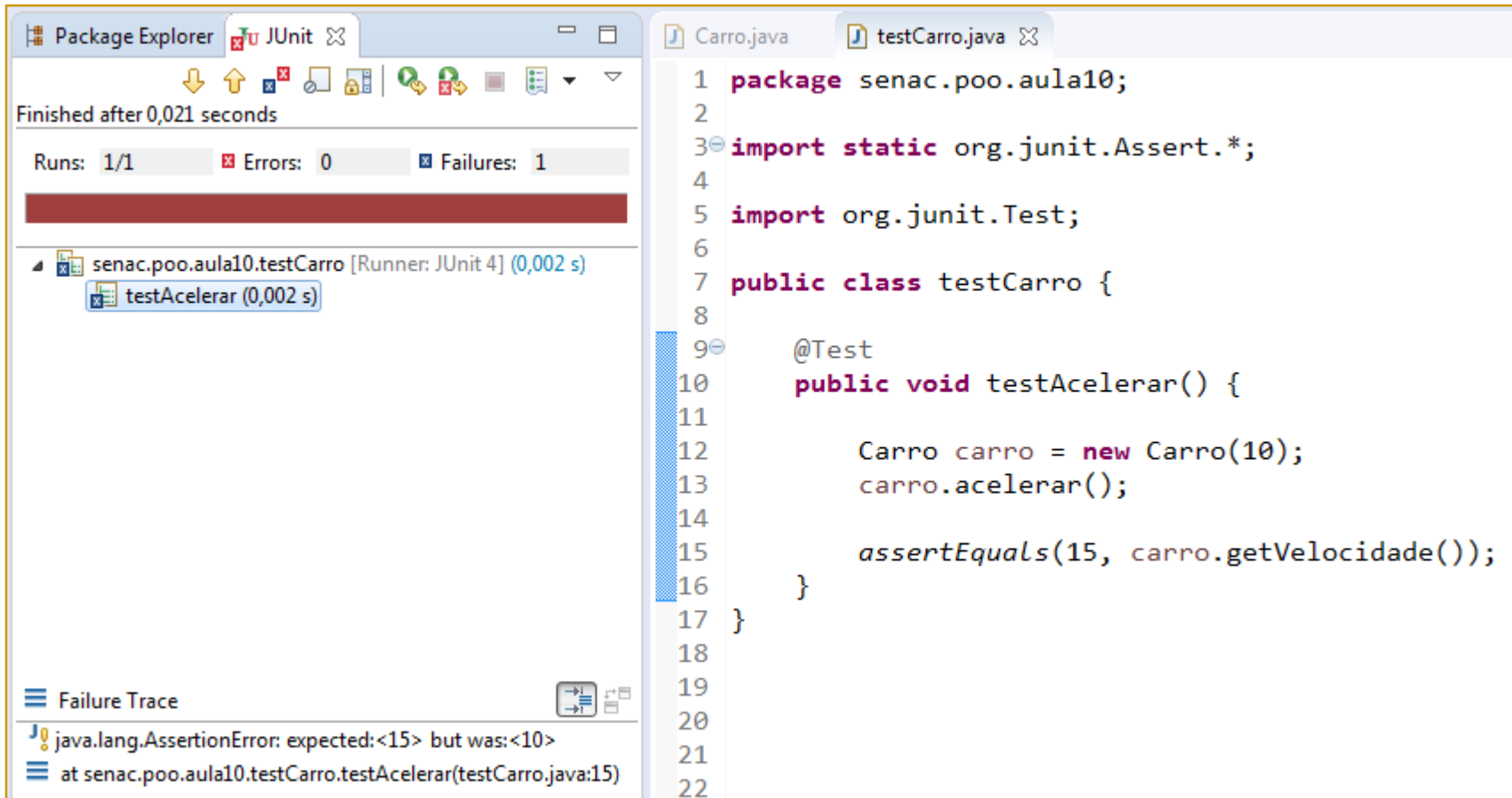
**JUnit Results (Left Panel):**

- Package Explorer: JUnit
- Finished after 0,014 seconds
- Runs: 1/1
- Errors: 0
- Failures: 0
- Test: senac.poo.aula10.testCarro [Runner: JUnit 4] (0,001 s)
- Failure Trace: (empty)

**Source Code (Right Panel):**

```
1 package senac.poo.aula10;
2
3 import static org.junit.Assert.*;
4
5 import org.junit.Test;
6
7 public class testCarro {
8
9     @Test
10     public void testAcelerar() {
11
12         Carro carro = new Carro(10);
13         carro.acelerar();
14
15         assertEquals(10, carro.getVelocidade());
16     }
17 }
18
19
20
21
```

# Exemplo



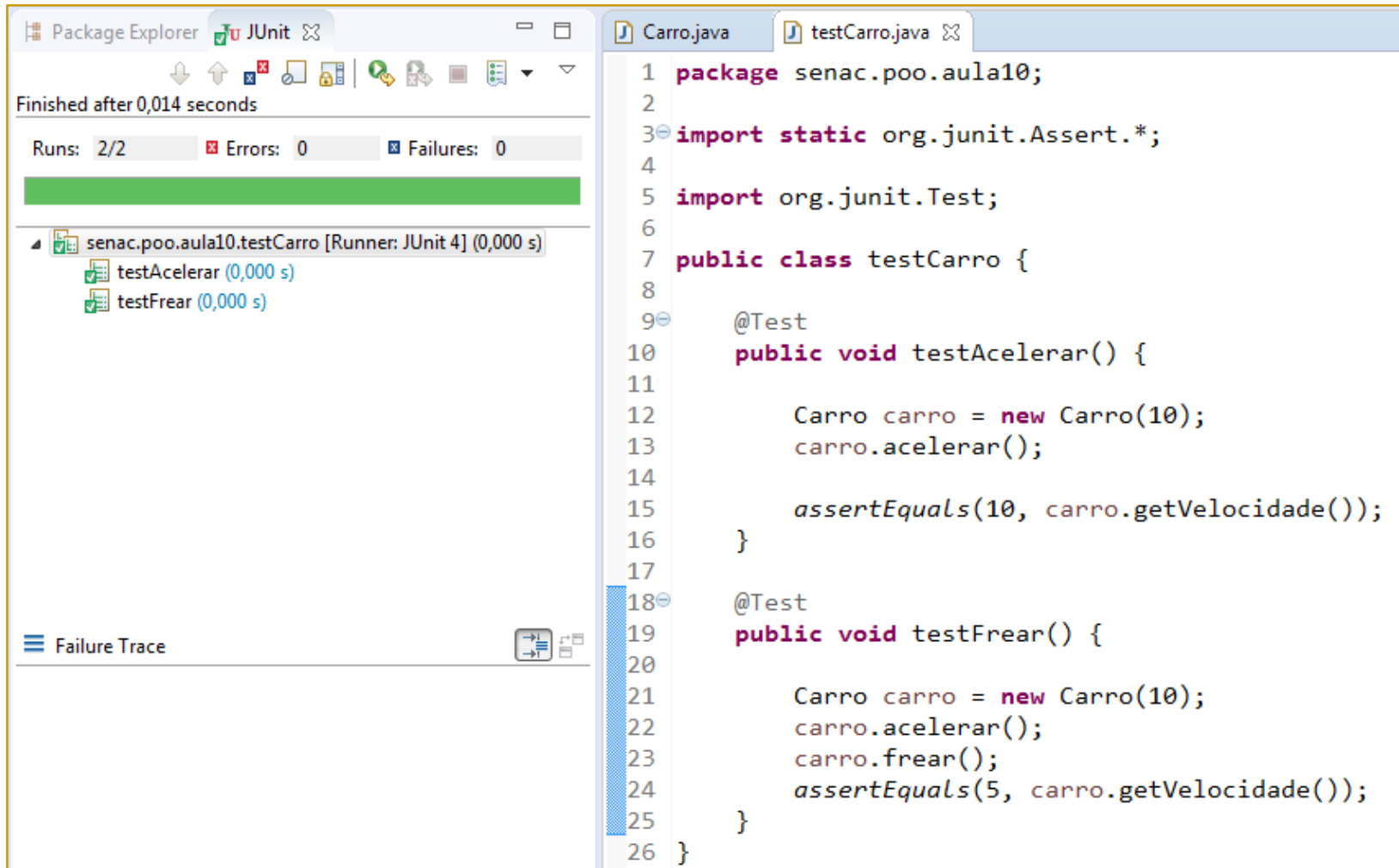
The screenshot displays an IDE with two tabs: 'Carro.java' and 'testCarro.java'. The 'testCarro.java' tab is active, showing the following code:

```
1 package senac.poo.aula10;
2
3 import static org.junit.Assert.*;
4
5 import org.junit.Test;
6
7 public class testCarro {
8
9     @Test
10     public void testAcelerar() {
11
12         Carro carro = new Carro(10);
13         carro.acelerar();
14
15         assertEquals(15, carro.getVelocidade());
16     }
17 }
```

The Package Explorer on the left shows the test suite 'senac.poo.aula10.testCarro' [Runner: JUnit 4] (0,002 s). It indicates 1 Run, 0 Errors, and 1 Failure. The failure is in the 'testAcelerar' method (0,002 s). The Failure Trace at the bottom shows:

```
java.lang.AssertionError: expected:<15> but was:<10>
    at senac.poo.aula10.testCarro.testAcelerar(testCarro.java:15)
```

# Exemplo



The screenshot displays an IDE interface with two main panels. The left panel shows the 'JUnit' test runner results, indicating a successful run of two tests: 'testAcelerar' and 'testFrear', both taking 0,000 seconds. The right panel shows the source code for 'testCarro.java'.

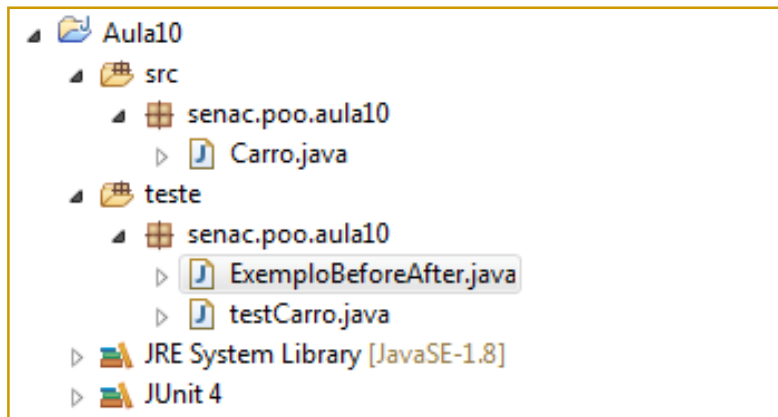
**JUnit Runner Results:**

- Finished after 0,014 seconds
- Runs: 2/2
- Errors: 0
- Failures: 0
- Test Results:
  - senac.poo.aula10.testCarro [Runner: JUnit 4] (0,000 s)
    - testAcelerar (0,000 s)
    - testFrear (0,000 s)

**Source Code (testCarro.java):**

```
1 package senac.poo.aula10;
2
3 import static org.junit.Assert.*;
4
5 import org.junit.Test;
6
7 public class testCarro {
8
9     @Test
10    public void testAcelerar() {
11
12        Carro carro = new Carro(10);
13        carro.acelerar();
14
15        assertEquals(10, carro.getVelocidade());
16    }
17
18    @Test
19    public void testFrear() {
20
21        Carro carro = new Carro(10);
22        carro.acelerar();
23        carro.frear();
24        assertEquals(5, carro.getVelocidade());
25    }
26 }
```

# Exemplo com Before e After



```
ExemploBeforeAfter.java
1 package senac.poo.aula10;
2
3 import static org.junit.Assert.*;
6
7 public class ExemploBeforeAfter {
8
9     @Test
10     public void test() {
11         fail("Not yet implemented");
12     }
13
14 }
```



# Exemplo com Before e After

```
ExemploBeforeAfter.java
1 package senac.poo.aula10;
2
3 import org.junit.After;
4 import org.junit.AfterClass;
5 import org.junit.Before;
6 import org.junit.BeforeClass;
7 import org.junit.Test;
8
9 public class ExemploBeforeAfter {
10
11     @BeforeClass
12     public static void metodoBeforeClass() {
13         System.out.println("BeforeClass");
14     }
15
16     @Before
17     public void metodoBefore() {
18         System.out.println("Before");
19     }
20
21     @Test
22     public void test1() {
23         System.out.println("Teste 1");
24     }
25 }
```

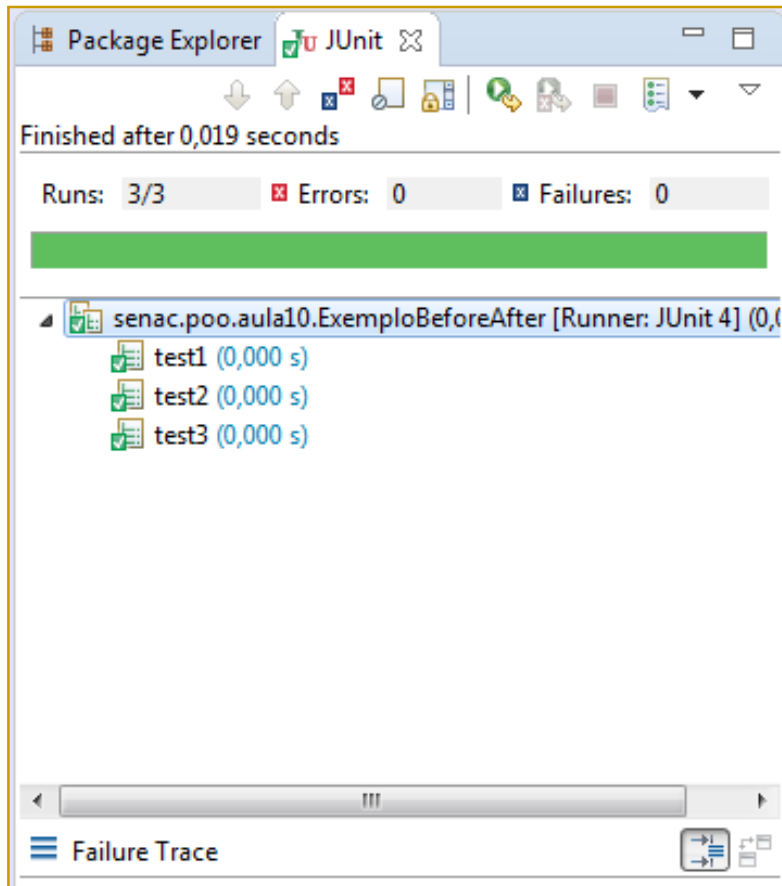
# Exemplo com Before e After

```
ExemploBeforeAfter.java ✕
20
21 @Test
22 public void test1() {
23     System.out.println("Teste 1");
24 }
25
26 @Test
27 public void test2() {
28     System.out.println("Teste 2");
29 }
30
31 @Test
32 public void test3() {
33     System.out.println("Teste 3");
34 }
35
36 @After
37 public void metodoAfter() {
38     System.out.println("After");
39 }
40
```

# Exemplo com Before e After

```
35
36 @After
37 public void metodoAfter() {
38     System.out.println("After");
39 }
40
41 @AfterClass
42 public static void metodoAfterClass() {
43     System.out.println("AfterClass");
44 }
45 }
46
```

# Exemplo com Before e After



```
BeforeClass  
Before  
Teste 1  
After  
Before  
Teste 2  
After  
Before  
Teste 3  
After  
AfterClass
```