# COMPARACION ENTRE RSPEC Y JUNIT5

- En Rspec se puede utilizar el **“describe”** y **“context”**, esto permite entender que es lo que se quiere probar brindando una mayor claridad
- En Junit no hay **“describe”** pero si ofrece el marcado **@test** para identificar que es un método de prueba además de **@DisplayName** para declarar un nombre al método de prueba

Rspec

```
describe StringCalculator do
  describe ".add" do
    context "given an empty string" do
      it "return zero" do
        expect(StringCalculator.add("")).to eq(0)
      end
    end
  end

  describe ".addTwo" do
    context "given '4'" do
      it "returns 4" do
        expect(StringCalculator.addTwo("4")).to eql(4)
      end
    end

    context "given '10'" do
      it "returns 10" do
        expect(StringCalculator.addTwo("10")).to eql(10)
      end
    end
  end
```

JUnit

```
@Test
void testMethod1() {
    System.out.println("Metodo test 1");
}

@DisplayName("Nombre de Test2")
@Test
void testMethod2() {
    System.out.println("Metodo test 2");
}

@Test
@Disabled("Metodo de test pendiente")
void testMethod3() {
    System.out.println("Metodo test 1");
}
```

Los Matchers que ofrece Rspec y Junit, ambos permiten verificar la igualdad de un valor esperado y un valor dado

```
14      it "returns 4" do
15        expect(StringCalculator.addTwo("4")).to eql(4)
16      end
```

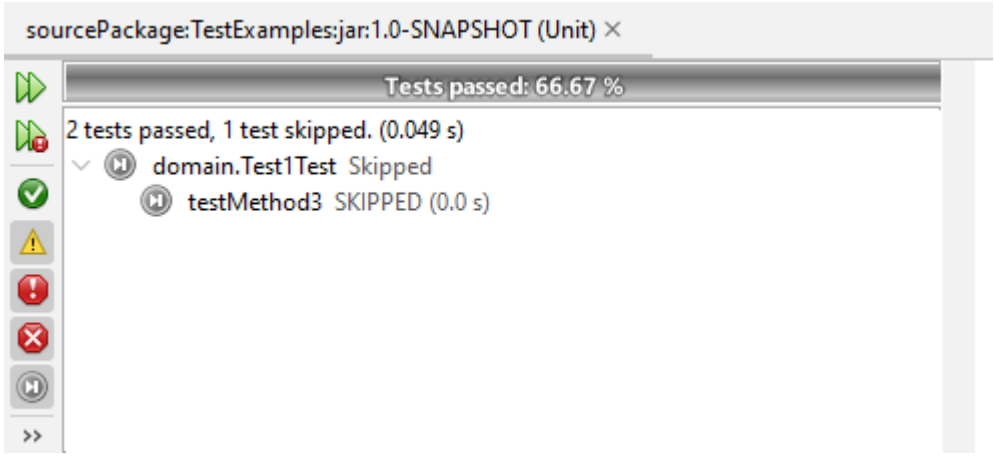
```
@Test
void testAssertEqual() {
    assertEquals("ABC", "ABC");
    assertEquals(20, 20, "optional assertion message");
    assertEquals(2 + 2, 4);
}
```

En JUNIT tenemos el marcado de **@Disable** que permite deshabilitar un test ya sea porque falta terminar la implementación o porque simplemente no queremos que se ejecute, esto en Rspec no es posible hacer a menos que se comente el test que no queremos que se corra:

En este ejemplo el método “testMethod3” está deshabilitado

```
33      }
34
35      @Test
36      @Disabled("Metodo de test pendiente")
37      void testMethod3() {
38          System.out.println("Metodo test 1");
39      }
40
```

Resultado: muestra que el test ha sido Skipped (omitido)



En ambos Framework nos permiten realizar testing anidados, en el caso de Rspec gracias al context puedo hacer mas de un test para una función en específico, en la imagen se está evaluando el método addTwo en dos situaciones, uno para evaluar que retorna 4 y otro 10, en el ejemplo de JUNIT usamos el marcado **@Nested** indica que la clase InnerMostClass es anidada de la clase InnerClass

```
describe ".addTwo" do
  context "given '4'" do
    it "returns 4" do
      expect(StringCalculator.addTwo("4")).to eql(4)
    end
  end

  context "given '10'" do
    it "returns 10" do
      expect(StringCalculator.addTwo("10")).to eql(10)
    end
  end
end
```

```
41      @Nested
        class InnerClass {
43
44          @BeforeEach
            void beforeEach() {
46              System.out.println("**--- InnerClass :: beforeEach :: Executed before each test method ---**");
47          }
48
49          @AfterEach
            void afterEach() {
51              System.out.println("**--- InnerClass :: afterEach :: Executed after each test method ---**");
52          }
53
54          @Test
            void testMethod1() {
56              System.out.println("**--- InnerClass :: testMethod1 :: Executed test method1 ---**");
57          }
58
            @Nested
            class InnerMostClass {
61
62              @BeforeEach
                void beforeEach() {
64                  System.out.println("**--- InnerMostClass :: beforeEach :: Executed before each test method ---**");
65              }
66
67              @AfterEach
                void afterEach() {
69                  System.out.println("**--- InnerMostClass :: afterEach :: Executed after each test method ---**");
70              }
71
72              @Test
                void testMethod2() {
74                  System.out.println("**--- InnerMostClass :: testMethod2 :: Executed test method2 ---**");
```