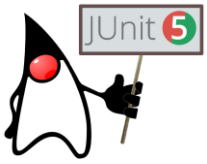


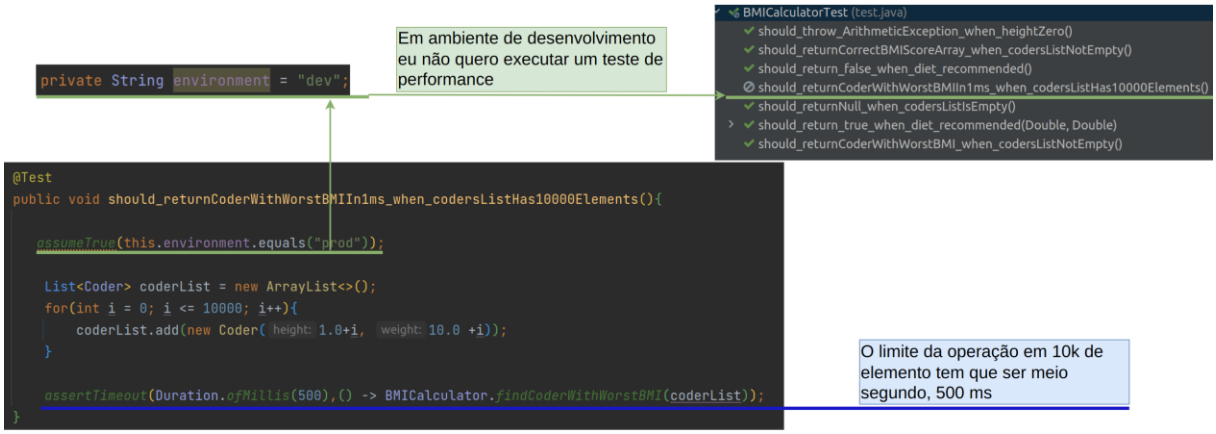
Testes parametrizáveis – Java

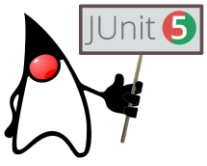
Geralmente escrevemos testes com valores fixos nos parâmetros, afinal de contas precisamos saber exatamente o que esperar. Mas pode ser interessante usar **diversos parâmetros em um teste**, por exemplo, usar **valores de extremidade** para validar mais ainda o teste:

Da pra **pegar parâmetros de arquivos csv** usando a anotação **@CsvFileSource** que recebe o caminho do arquivo como valor, além de poder fazer algumas operações de linha, tipo skipar.

Teste de performance e Assumptions – Java

JUnit tem uma **assertiva bem útil para testar a performance de um método, testes de performance** são geralmente executados em ambientes diferentes do de dev e por isso o Junit tem **uma "assumption" para ajudar a especificar ambientes em que testes devem ou não ser executados:**





Nested classes – Java

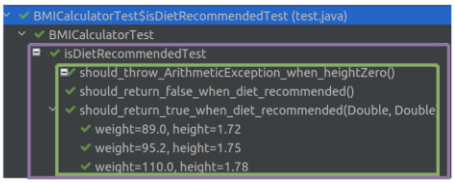
São **classes internas que podem agrupar um montante de teste**, é usado de forma semântica:

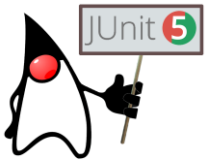
```java
@Nested
class isDietRecommendedTest{

    @ParameterizedTest(name = "weight={0}, height={1}")
    @CsvSource(value = {"89.0, 1.72", "95.2, 1.75" , "110.0, 1.78"} )
    public void should_return_true_when_diet_recommended(Double codersWeight, Double codersHeight){...}

    @Test
    public void should_return_false_when_diet_recommended(){...}

    @Test
    public void should_throw_ArithmeticException_when_heightZero(){...}
}
```





Disable – Java

Serve **para desabilitar algum teste**, é interessante que podem ter condições específicas para desabilitar um teste, como por exemplo um SO diferente:

```java
@Test
@Disabled
public void should_returnCoderWithWorstBMI_when_codersListNotEmpty(){...}
```

```java
@Test
@DisabledOnOs(OS.LINUX)
public void should_returnCoderWithWorstBMI_when_codersListNotEmpty(){...}
```