Relatório do curso de TEST DRIVEN DEVELOPMMENT (TDD)
ITA (Instituto Tecnológico de Aeronaútica)

Quebra de Strings com CamelCase

Aluno : **Tiago Ribeiro Santos**

Email : tiago.programador@hotmail.com

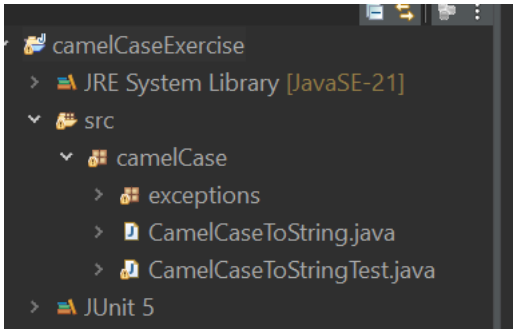
Linkedin: https://www.linkedin.com/in/tiagoribeirosantos/

Descrição de como ocorreu os testes em passos(baby-steps)

1° Passo – Criei o cenário para realização dos testes, em nosso caso, **o cenário de quebra de strings com CamelCase**(convenção de escrita de palavras compostas ou frases em que cada palavra é iniciada com maiúsculas e unidas sem espaço).

* Foi utilizado em todo desenvolvimento , assim como os testes unitários o framework **Junit5.**

2º Passo – Criar o projeto denominado "**camelCaseExercise**", a classe de produção chamada de "**CamelCaseToString**" (para nossas regras para conversão de Strings para o formato camelCase), assim como as classe para realização de nossos testes unitários.



3º Passo – Criação da Classe de Produção, **public class CamelCaseToString** que será utilizada para aplicarmos toda a nossa lógicas de entrada e saída que são esperadas no console para o usuário em formato CamelCase.

* Criei variáveis constantes com Regex (expressões regulares) para verificações lógicas em nosso método, **public static List<String> converterCamelCase(String original);**

```
public static final String START_WITH_NUMBER = "^[0-9].*";
public static final String HAS_SPECIAL_CHARACTER = "[^a-zA-Z0-9]+";
public static final String SEPARATE_WORDS = "(?<!(^|[A-Z0-9]))(?=[A-Z0-9])|(?<!(^|[^A-Z]))(?=[0-9])|(?<!(^|[^0-9]))(?=[A-Za-z])|(?<!^
```

* De acordo com o que é esperado como exemplos de entrada e saída,podemos citar alguns dos **cenários de falha(na qual eu elaborei a minha validação para identificar tais cenários que não são esperados)**:

Primeiro Cenário – Cenário na qual uma String se inicia com um número (Ocorre uma Exception específica que criei,pois uma string não pode começar com números.

```
if(original.matches(START_WITH_NUMBER)) {
    throw new CamelCaseException(INVALID_START_ERROR);
}
```

Segundo Cenário – Cenário na qual uma String contém caracteres especiais embutidos(Na qual não pode ocorrer também):

```
if(original.matches(HAS_SPECIAL_CHARACTER)) {
    throw new CamelCaseException(INVALID_CHARACTER_MESSAGE);
}
```

Terceiro Cenário – Cenário na qual identifica se uma string está vazia ou nula, através do método **isValidString(String original)** da qual criei pensando nessa lógica de validação.

```
if(!isValidString(original)) {
    throw new CamelCaseException(INVALID_STRING);
}
```

Se por acaso, o método **public static List<String> converterCamelCase(String original),** não conter nenhum dos cenários "falhos", é realizado a conversão da string informada como (parâmetro) em minísculas e separando frases Compostas em várias palavras, assim finalmente convertendo-as em formato **"camelCase".**

Foram criados também os seguintes métodos, para testar a nossa string conforme seguintes condições.

3.1 – método **isNumberBetweenString (str) ->** Existe números entre letras? **(retorna true or false)**

3.2 – método **isValidString(str) ->** A string é válida? É uma string diferente de nulo ou maior que 0?

Além do mais, criei uma classe de Exception genérica para todas as exceções disparadas (caso a string estiver vazia ou existir caracteres especiais) denominada de **“CamelCaseException”**

```java
public class CamelCaseException extends RuntimeException {

    public CamelCaseException(String message) {
        super(message);
    }
}
```

4º Passo – Após toda lógica presente para validação de strings válidas, assim como a classe de exception criadas , criei minha classe de teste chamada de **CamelCaseToStringTest.java** 

5º Passo – Após a criação do CamelCaseToStringTest(a nossa classe de Teste), eu havia criado um teste específico para testar cada exceção (como exemplo : string começa com número, possui caracter especial ,se a string é vazia ou nula).

```java
    @Test
    public void testIfExistsEmptyStr() {
        camelCase.converterCamelCase("");
    }
```

(Teste Caso se a string é vazia)

```java
    @Test
    public void testIfStrStartNumber() {
        camelCase.converterCamelCase("10Primeiros");
    }
```

(Teste Caso a string começa com números)

Em seguida após uma análise de código acabei refatorando e realizando todos os possivéis cenários de “exceção” em único teste da qual chamei esse teste de **testExpectedExceptions()** da qual é esperado a classe de Exceção **CamelCaseException.class**

```
@Test(expected = CamelCaseException.class)
public void testExpectedExceptions() {
    camelCase.converterCamelCase("");
    camelCase.converterCamelCase(" ");
    camelCase.converterCamelCase("10Primeiros");
    camelCase.converterCamelCase("nome#Composto");
}
```

Por fim, criei mais 3 métodos de testes para verificar se toda a string(após a conversão para o formato camelCase),possui exatamente os indíces dessa List ,outro teste para verificar se a string é nula e mais um outro teste para validar se existe números entre strings,esperados através das **Assertivas do Junit5(Asserts)**

```
@Test
public void testCamelCase() {
    //verify if string list index has 1
    strList = camelCase.converterCamelCase("nome");
    Assert.assertEquals(strList.size(),1);
    Assert.assertEquals(strList.get(0),"nome");

    //verify if string exists camelCase
    strList = camelCase.converterCamelCase("nomeComposto");
    Assert.assertEquals(strList.size(), 2);
    Assert.assertEquals(strList.get(0), "nome");
    Assert.assertEquals(strList.get(1), "composto");

    //verify if string exists camelCase
    strList = camelCase.converterCamelCase("nomeComposto");
    Assert.assertEquals(strList.size(), 2);
    Assert.assertEquals(strList.get(0), "nome");
    Assert.assertEquals(strList.get(1), "composto");

    //verify string CPF
    strList = camelCase.converterCamelCase("CPF");
    Assert.assertEquals(strList.size(), 1);
    Assert.assertEquals(strList.get(0), "CPF");

    //verify string recupera10primeiros
    strList = camelCase.converterCamelCase("recupera10Primeiros");
    Assert.assertEquals(strList.size(), 3);
    Assert.assertEquals(strList.get(0), "recupera");
```

```java
@Test
public void testIfStrNull() {
    Assert.assertFalse(camelCase.isValidString(null));
}
```

```java
@Test
public void testIfExistsNumberBetweenString() {
    Assert.assertTrue(camelCase.isNumberBetweenString("recupera10Primeiros"));
    Assert.assertTrue(camelCase.isNumberBetweenString("abdcdeseffd123asbc"));
}

//verify if string size is as expected
@Test
public void testStrSize() {
    Assert.assertEquals(9, "numeroCPF".length());
}
```