Camel Case TDD

Modelo de Apresentação

TesteUso do Teste:  
Código antes do teste (Apenas partes modificadas):  
  
Código depois do teste:  
  
Conclusão

Contents

[iniciaClasse 2](#_Toc122387472)

[retornaLista 3](#_Toc122387474)

[conteudoLista 4](#_Toc122387476)

[listaDeStrings 5](#_Toc122387478)

[separaPalavras 6](#_Toc122387480)

[retornaPrimeiraMinuscula 7](#_Toc122387482)

[retornaPalavraMaiuscula 8](#_Toc122387484)

[separaPalavraMaiuscula 9](#_Toc122387486)

[umCharEMesmoTipo 10](#_Toc122387488)

[separaNumeros 11](#_Toc122387490)

[erroSeComecarNumero 12](#_Toc122387492)

[erroSeTiverEspecial 13](#_Toc122387494)

[converterCamelCase 14](#_Toc122387496)

----------------------------------------------------------------------------------------------------------------------------

# iniciaClasse

Uso do Teste: Para ver se a classe que contém as funções para criar a lista de palavras existe e se está iniciando normalmente.

Código antes do teste:

Código depois do teste:

**public class** CamelCase {

}

Conclusão:

A solução mais direta foi criar uma classe vazia chamada CamelCase.

----------------------------------------------------------------------------------------------------------------------------

# retornaLista

Uso do Teste: O método “converterCamelCase” da classe “CamelCase” deve retornar uma lista.

Código antes do teste:

**public class** CamelCase {

}

Código depois do teste:

**public class** CamelCase {

**public** List converterCamelCase(String string) {

List l = new ArrayList();

return l;

}

}

Conclusão:

Foi criada a função converterCamelCase, que recebe uma string como argumento (De acordo com as exigencias do projeto) e deve retornar uma lista.

# conteudoLista

Uso do Teste: Teste Para ver se a lista que a função “converterCamelCase” retorna não está vazia

Código antes do teste:

**public class** CamelCase {

**public** List converterCamelCase(String string) {

List l = new ArrayList();

return l;

}

}

Código depois do teste:

**public class** CamelCase {

**public** List converterCamelCase(String string) {

List l = new ArrayList();

l.add(1);

return l;

}

}

Conclusão:

Agora a função converterCamelCase retorna uma Lista com um único número, o que supre as exigências do teste.

----------------------------------------------------------------------------------------------------------------------------

# listaDeStrings

Uso do Teste: Ver se a lista criada é composta por apenas strings.

Código antes do teste:

**public class** CamelCase {

**public** List converterCamelCase(String string) {

List l = new ArrayList();

l.add(1);

return l;

}

Código depois do teste:

**public class** CamelCase {

**public** List converterCamelCase(String string) {

List l = new ArrayList();

l.add(string);

return l;

}

}

Conclusão:

Adicionando a string já existente à lista, a função retornará apenas strings.

----------------------------------------------------------------------------------------------------------------------------

# separaPalavras

Uso do Teste: A função "separaMaiuscula" deve as palavras separadas pelas suas letras maiúsculas.

Código antes do teste:

**public class** CamelCase {

**public** List converterCamelCase(String string) {

List l = new ArrayList();

l.add(string);

return l;

}

Código depois do teste:

**public class** CamelCase {

**public** List converterCamelCase(String string) {

List l = new ArrayList();

l.add(string);

return l;

}

**public** String[] separaMaiuscula(String string) {

String[] palavras = string.split("(?=\\p{Lu})");

return palavras;

}

}

}

Conclusão:

Criamos a função que retorna uma array de palavras separadas por letras maiúsculas usando Regex.

----------------------------------------------------------------------------------------------------------------------------

# retornaPrimeiraMinuscula

Uso do Teste: A função "formatarPalavra" deve retornar palavras que comecem com letra minuscula ou maiuscula como totalmente minusculas.

Código antes do teste:

**public** String[] separaMaiuscula(String string) {

String[] palavras = string.split("(?=\\p{Lu})");

return palavras;

}

Código depois do teste:

**public** String[] separaMaiuscula(String string) {

String[] palavras = string.split("(?=\\p{Lu})");

**return** palavras;

}

**public** String formatarPalavra(String string) {

**return** string.substring(0,1).toLowerCase() + string.substring(1);

}

Conclusão:

Criamos a função formatarPalavra, que retorna uma string diminuindo a primeira letra de uma palavra.

----------------------------------------------------------------------------------------------------------------------------

# retornaPalavraMaiuscula

Uso do teste: A função "formatarPalavra" deve retornar palavras que sejam completamente Maiusculas, sem nenhuma modificação

Código antes do teste:

**public** String formatarPalavra(String string) {

**return** string.substring(0,1).toLowerCase() + string.substring(1);

}

Código depois do teste:

**public** String formatarPalavra(String string) {

**for**(int i=0; i < string.length(); i++){

**if**( !Character.*isUpperCase*( string.charAt(i)))

**return** string.substring(0,1).*toLowerCase()* + string.substring(1);

}

return string;

}

}

Conclusão:

A função passa por todos os caracteres da string, e caso um seja em caixa baixa, retorna com a primeira letra em caixa baixa (Não leva em consideração palavras que tenham letras maiúsculas como primeiras letras, como IMPossivel, pois estariam fora do caso de camelcase de qualquer jeito). Caso contrário, retorna a palavra como é.

----------------------------------------------------------------------------------------------------------------------------

# separaPalavraMaiuscula

Uso do teste: A função "separaMaiuscula" retornar, como palavras separadas, 3 ou mais caracteres maiúsculos que estejam juntos.

Código antes do teste:

**public** String[] separaMaiuscula(String string) {

String[] palavras = string.split("(?=\\p{Lu})");

**return** palavras;

}

Código depois do teste:

**public** String[] separaMaiuscula(String string) {

String[] palavras = string.split("(?=\\p{Lu})");

List<String> subarray = **new** ArrayList<>();

**for** (int i = 0; i < palavras.length; i++)

{

**if** (palavras[i].length() < 2)

{

**while**((i < palavras.length-1) && (palavras[i+1].length() <2))

{   
palavras[i+1] = palavras[i] + palavras[i+1];  
i++;  
}

}

subarray.add(palavras[i]);

}

**return** subarray.*toArray*(**new** String[0]);

}

Conclusão:

Antes de retornar a lista, a função vai analisar as palavras que foram separadas. Aquelas que tiverem o tamanho menor que 2 (Forem de apenas um caracter de tamanho), ou seja, forem apenas maiúsculas, vão ser unidas para formar uma palavra, até que seja encontrada uma maiúscula seguida de minúscula.

----------------------------------------------------------------------------------------------------------------------------

# umCharEMesmoTipo

Uso do teste: A função "charETipo" deve retornar verdadeiro caso a string seja apenas um caractere de tamanho e do mesmo tipo que o indicado (Numerico ou não).

Código antes do teste:

**return** l;

}

**public** String[] separaMaiuscula(String string) {

String[] palavras = string.split("(?=\\p{Lu})");

Código depois do teste:

**return** l;

}

**public** **boolean** charETipo(String string, **boolean** isNumeric) {

**return** (string.length() < 2) && (isNumeric == Character.*isDigit*(string.charAt(0)));

}

**public** String[] separaMaiuscula(String string) {

String[] palavras = string.split("(?=\\p{Lu})");

Conclusão:

Criamos a função que retorna true caso a string seja de um caracter e do tipo especificado, assim transformando esses dois testes em um para funções futuras.

----------------------------------------------------------------------------------------------------------------------------

# separaNumeros

Uso do teste: A função "separaMaiuscula" deve retornar os numeros separados como palavras.

Código antes do teste:

**public** String[] separaMaiuscula(String string) {

String[] palavras = string.split("(?=\\p{Lu})");

List<String> subarray = **new** ArrayList<>();

**for** (int i = 0; i < palavras.length; i++)

{

**if** (palavras[i].length() < 2)

{

**while**((i < palavras.length-1) && (palavras[i+1].length() <2))

{   
palavras[i+1] = palavras[i] + palavras[i+1];  
i++;  
}

}

subarray.add(palavras[i]);

}

**return** subarray.*toArray*(**new** String[0]);

}

Código depois do teste:

**public** String[] separaMaiuscula(String string) {

String[] palavras = string.split("(?=[\\p{Lu}\\d])");

List<String> subarray = **new** ArrayList<>();

**for** (int i = 0; i < palavras.length; i++)

{

**if** (palavras[i].length() < 2)

{

**boolean** isNumeric = Character.*isDigit*(palavras[i].charAt(0));

**while**((i < palavras.length-1) && charETipo(palavras[i+1],isNumeric))

{   
palavras[i+1] = palavras[i] + palavras[i+1];  
i++;  
}

}

subarray.add(palavras[i]);

}

**return** subarray.*toArray*(**new** String[0]);

}

Conclusão:

Agora a função também separa caracteres numéricos como se fossem letras maiúsculas, que depois são unidas em uma palavra só. E diferencia os dois usando a variável isNumeric (Caso contrário iria juntar numeros com letras maiúsculas).  
----------------------------------------------------------------------------------------------------------------------------

# erroSeComecarNumero

Uso do teste: A função “converterCamelCase” deve dar erro caso a string comece com um caracter numerico.

Código antes do teste:

**public** List converterCamelCase(String string) {

List l = **new** ArrayList();

l.add(string);

return l;

}

Código depois do teste:

CamelCase.java:

**public** List converterCamelCase(String string) {

List l = **new** ArrayList();

if (Character.*isDigit*(string.charAt(0)))

**throw** **new** FormatacaoErrada("String começa com caracter numérico");

l.add(string);

return l;

}

FormatacaoErrada.java (NOVO)

**public class** FormatacaoErrada **extends** RuntimeException {

**public** FormatacaoErrada(String message)

{

**super**(message);

}

**private static final long** *serialVersionUID* = 1L;

}

Conclusão:

A classe de erro foi criada e foi adicionada uma checagem na função converterCamelCase para lançar esse erro.  
----------------------------------------------------------------------------------------------------------------------------

# erroSeTiverEspecial

Uso do teste: A função converterCamelCase deve dar erro caso a string tenha algum caracter especial.

Código antes do teste:

**public** List converterCamelCase(String string) {

List l = **new** ArrayList();

if (Character.*isDigit*(string.charAt(0)))

**throw** **new** FormatacaoErrada("String começa com caracter numérico");

l.add(string);

return l;

}

Código depois do teste:

**public** List converterCamelCase(String string) {

List l = **new** ArrayList();

if (Character.*isDigit*(string.charAt(0)))

**throw** **new** FormatacaoErrada("String começa com caracter numérico");

if(string.matches(".\*[^A-Za-z0-9].\*"))

**throw new** FormatacaoErrada("String tem caracteres não alfanuméricos");

l.add(string);

return l;

}

Conclusão:

Foi adicionada uma checagem na função “converterCamelCase” para lançar o erro caso a string contenha algum caracter que não seja alfanumerico. O teste para saber é feito com um regex negativo que engobla todos os caracteres entre A e Z, 0 e 9.

----------------------------------------------------------------------------------------------------------------------------

# converterCamelCase

Uso do teste: A função converterCamelCase deve retornar a lista de strings como antes, mas com palavras que sejam separadas de uma string em camelcase.

Código antes do teste:

**public** List converterCamelCase(String string) {

List l = **new** ArrayList();

if (Character.*isDigit*(string.charAt(0)))

**throw** **new** FormatacaoErrada("String começa com caracter numérico");

if(string.matches(".\*[^A-Za-z0-9].\*"))

**throw new** FormatacaoErrada("String tem caracteres não alfanuméricos");

l.add(string);

return l;

}

Código depois do teste:

**public** List converterCamelCase(String string) {

List l = **new** ArrayList();

if (Character.*isDigit*(string.charAt(0)))

**throw** **new** FormatacaoErrada("String começa com caracter numérico");

if(string.matches(".\*[^A-Za-z0-9].\*"))

**throw new** FormatacaoErrada("String tem caracteres não alfanuméricos");

String[] palavras = separaMaiuscula(string);

**for** (String p : palavras)

{

l.add(formatarPalavra(p));

}

return l;

}

Conclusão:

Adicionando os métodos anteriormente criados ao método principal, todos devidamente testados previamente, podemos fazer com que o método cumpra a função esperada, separando a string e formatando as strings consequentes, dentro do limite de 10 linhas, e sem perder as funcionalidades anteriores.