Camel Case TDD

Modelo de Apresentação

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

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

# 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[] separaMaiscula(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 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[] separaMaiscula(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.

Antes:

public String[] separaMaiscula(String string) {

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

return palavras;

}

Depois:

public String[] separaMaiscula(String string) {

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

return palavras;

}

public String formatarPalavra(String string) {

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

}

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

Teste "retornaPalavraMaiscula" para a função "formatarPalavra" retornar palavras que sejam completamente maisculas iguais

Antes:

public String formatarPalavra(String string) {

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

}

Depois:

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;

}

Agora a função vai checar se algum caractere daquela palavra é em letras minúsculas, e se nenhum for, só retorna a string.

Teste "separaPalavraMaiscula" para a função "separaMaiuscula" retornar como palavras 3 ou mais caracteres que estejam todos em maiúsculo

Antes:

public String[] separaMaiscula(String string) {

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

return palavras;

}

Depois:

public String[] separaMaiscula(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]);

}

Agora a função vai checar por letras maiusculas separadas e uni-las para fazer palavras inteiras

Teste "umCharEMesmoTipo" para a função "charETipo" retornar verdadeiro caso a string seja apenas um caractere e do mesmo tipo que o indicado (Numerico ou não)

Antes:

return l;

}

Depois:

return l;

}

public boolean charETipo(String string, boolean isNumeric) {

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

}

public String[] separaMaiscula(String string) {

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

Criamos a função que retorna true caso a string seja de um caracter e do tipo especificado

Teste "separaNumeros" para a função "separaMaiscula" retornar os numeros separados como palavras

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

Antes:

public String[] separaMaiscula(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]);

}

Depois:

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]);

Agora a função também separa caracteres numéricos como se fossem letras maiúsculas, que depois são unidas em uma palavra só

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

Teste "erroSeComecarNumero" para a função converterCamelCase dar erro caso a string comece com um caracter numerico

Antes:

public List converterCamelCase(String string) {

List l = new ArrayList();

l.add(string);

return l;

}

Depois:

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;

}

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

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

Teste "erroSeTiverEspecial" para a função converterCamelCase dar erro caso a string tenha caracter especial

Antes:

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;

}

Depois:

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;

}

Foi adicionada uma checagem na função converterCamelCase para lançar o erro caso a string contenha algum caracter que não seja alfanumerico

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

Teste "converteCamelCase" para a função converterCamelCase retornar lista de palavras separadas de uma string em camelcase

Antes:

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;

}

Depois:

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 = separaMaiscula(string);

for (String p : palavras)

{

l.add(formatarPalavra(p));

}

return l;

}

Finalmente introduzimos os outros métodos a função principal