# *Processo de TDD utilizado*

Criar utilizando TDD um método que transforma uma cadeia de caracteres em camel case (http://pt.wikipedia.org/wiki/CamelCase) em uma lista de Strings com as palavras. O método deve possuir a seguinte assinatura: **"public static List<String> converterCamelCase(String original)".**

Para iniciar o projeto de Conversão **CamelCase** é criado o mapeamento da classe a partir de um primeiro teste simples com o objetivo de montar uma estrutura dos métodos iniciais.

Teste com uma palavra que tem todas as letras minúsculas, "nome".

1º Teste

Uma palavra com primeira letra minúscula

```
@Test
public void testConverterUmaPalavraComPrimeiraLetraMinuscula() {

	List<String> resultadoEsperado = Arrays.asList("nome");
	String camelCase = "nome";

	assertEquals(resultadoEsperado,
	Conversor.converterCamelCase(camelCase));

}
```

A classe Conversor não existe e seu método *converterCamelCase*(String original)) também não. Dessa forma, a classe foi criada que retorna apenas String original sem tratamento algum.

```
public class Conversor {

    private static List<String> listaPalavras;


    public static List<String> converterCamelCase(String original){

      Conversor converter = new Conversor();
      listaPalavra.add(original);

      return listaPalavras;
}
```

2º Teste

Uma palavra com primeira letra maiúscula

O segundo teste com a mesma palavra anterior, porém com letra inicial maiúscula.
Resulta em um erro, devido ser esperado que a letras inicial fiquem na forma minúscula. Conforme regra, dessa forma o código foi refatorado para que a palavra se tornem minúscula.

```
@Test
public void testConverterUmaPalavraComPrimeiraLetraMaiuscula() {

   List<String> resultadoEsperado = Arrays.asList("nome");

   String camelCase = "Nome";

   assertEquals(resultadoEsperado, Conversor.converterCamelCase(camelCase));

}
```

Apenas alterei o código da classe Conversor para que fique minúscula qualquer palavra a priori, através do método de String toUpperCase() no parâmetro original do método converterCamelCase .

```
public class Conversor {

    private static List<String> listaPalavras;


    public static List<String> converterCamelCase(String original){

    Conversor converter = new Conversor();

    listaPalavra.add(original.lowerCase());

    return listaPalavras;
}
```

3º Teste

Separar palavras compostas (todas minúsculas)

Gerado novo erro devido o código não separar palavras compostas, apenas torna minúsculas.

A palavra composta deve ser quebrada e em seguida transformada em minúsculas.

```
@Test
public void testConverterDuasPalavrasCompostas() {

   List<String> resultadoEsperado = Arrays.asList("nome", "composto");
   String camelCase = "nomeComposto";

   assertEquals(resultadoEsperado, Conversor.converterCamelCase(camelCase));

}


public class Conversor {

    private static List<String> listaPalavras;


    public static List<String> converterCamelCase(String original){

    Conversor converter = new Conversor();

    listaPalavra.add(original.lowerCase());

    return listaPalavras;
}
```

Criado novos métodos de tratamento da lista.

- List<String> **desfazerUmCamelCase** (String original)
- List<String> **tornaListaMinuscula**(List<String> listaPalavras)
- Pattern **extrairPadrao**(Pattern padrao)

```java
public static List<String> desfazerUmCamelCaseEmLista(String original){

   Pattern padrao = null;

   List<String> listaTratada;

   listaTratada = Arrays.asList(extrairPadrao(padrao).split(original));

   listaTratada = tornaListaMinuscula(listaTratada);

   return listaTratada;

}
```

O método abaixo auxilia o desfazerUmCamelCaseEmLista quebrando a String composta quando houver letras maiúsculas no meio da palavra.

```java
private static Pattern extrairPadrao(Pattern padrao) {
   padrao = Pattern.compile("(?=[A-Z])");
   return padrao;
}
```

O método abaixo auxilia o desfazerUmCamelCaseEmLista convertendo as palavras em minúscula.

```java
private static List<String> tornaListaMinuscula(List<String> listaPalavras) {

   List<String> novaListaPalavras = new ArrayList<String>();


   for (int i =0; i < listaPalavras.size();i++){

       novaListaPalavras.add(listaPalavras.get(i).toLowerCase());
  }

  return novaListaPalavras;

}
```

4º Teste

Separar palavras compostas (primeira Maiúsculas)

Separar palavras compostas, sendo que a primeira palavra nesse teste é maiúscula.
A rotina gerou erro causando uma terceira string vazia na lista.  A solução criada foi um tratamento para remover strings vazias quando for quando adicionada a lista.
Nesse tratamento foi criado um método de verificação

```
@Test
public void testConverterDuasPalavrasCompostasEmMaiusculas() {

   List<String> resultadoEsperado = Arrays.asList("nome", "composto");
   String camelCase = "NomeComposto";

   assertEquals(resultadoEsperado, Conversor.converterCamelCase(camelCase));

}
```

O método abaixo auxilia o desfazerUmCamelCaseEmLista convertendo as palavras em minúscula. Adicionado o tratamento para verifica uma String vazia na lista.

```
private static List<String> tornaListaMinuscula(List<String> listaPalavras) {

   List<String> novaListaPalavras = new ArrayList<String>();


   for (int i =0; i < listaPalavras.size();i++){

      // Verificando se tem String vazia abaixo
      if (!listaPalavras.get(i).isEmpty()) {
         novaListaPalavras.add(listaPalavras.get(i).toLowerCase());
      }
  }

  return novaListaPalavras;

}
```

5º Teste

SIGLAS

Gerou erro quando separou a sigla CPF no teste abaixo, criando uma lista de Strings com apenas letras minúscula.

```
@Test
public void testConverterSiglaMaiuscula() {

  List<String> resultadoEsperado = Arrays.asList("CPF");

  String camelCase = "CPF";

  assertEquals(resultadoEsperado, Conversor.converterCamelCase(camelCase));

}
```

Refatorando o código para tratar as Siglas.

```
private static List<String> tornaListaMinuscula(List<String> listaPalavras) {

   List<String> novaListaPalavras = new ArrayList<String>();
   String guardaSigla = "";

   for (int i =0; i < listaPalavras.size();i++){

      if (!listaPalavras.get(i).isEmpty()) {

         if (possuiApenasUmaLetra(listaPalavras.get(i)))
            guardaSigla += listaPalavras.get(i);

             if (!guardaSigla.isEmpty()){

       if !possuiApenasUmaLetra(listaPalavras.get(i))||(listaPalavras.size()
      - 1==i)){

          novaListaPalavras.add(guardaSigla);
          guardaSigla = "";

         }
       }

          if (!possuiApenasUmaLetra(listaPalavras.get(i)))
              novaListaPalavras.add(listaPalavras.get(i).toLowerCase());
            }
         }

         return novaListaPalavras;

   }
```

6º Teste

Separar lista de palavra com SIGLA

Sigla e palavras. Não gerou erro algum, portanto, não houve refatoramento.

```
@Test
public void testConverterSiglaCompostaComPalavra() {

   List<String> resultadoEsperado = Arrays.asList("numero", "CPF");
   String camelCase = "numeroCPF";

 assertEquals(resultadoEsperado,    Conversor.converterCamelCase(camelCase));

 }

}
```

7º Teste

Sigla e palavras compostas

Separou, entretanto invertou as ordem, gerando erro no resultado esperado.
Refatorado a classe principal Converte.

```
@Test
public void testConverterSiglaCompostaComDuasPalavras() {

   List<String> resultadoEsperado = Arrays.asList("numero", "CPF",
"contribuinte");

   String camelCase = "numeroCPFContribuinte";

   assertEquals(resultadoEsperado, Conversor.converterCamelCase(camelCase));

}
```

8º Teste

Numeração entre palavras

Refatorar a rotina do método Pattern extrairPadrao(Pattern padrao) para permitir separar quando encontrar números na String.

```
@Test
public void testConverterComNumeracao() {

   List<String> resultadoEsperado = Arrays.asList("recupera", "10",
   "primeiros");

   String camelCase = "recupera10Primeiros";

   assertEquals(resultadoEsperado, Conversor.converterCamelCase(camelCase));

}

private static Pattern extrairPadrao(Pattern padrao) {
      padrao = Pattern.compile("(?=[A-Z])|(?<=\\D)(?=\\d)");
      return padrao;
}
```

9º Teste

Criar tratamento com exceção para evitar número no inicio da String

```
/**
* 10Primeiros - Inválido - Não deve começar com números
*
*/

@Test(expected=ComecaComNumerosException.class)

public void comecaComNumerosException() {

   String camelCase = "10Primeiros";

   Conversor.converterCamelCase(camelCase);

}
```

Rotina criada na classe Converte

```
private boolean comecaComNumero(String original){
   return Character.isDigit(original.charAt(0));

}


package tddcamelcase;

public class ComecaComNumerosException extends RuntimeException {

   public ComecaComNumerosException(String msg) {
     super(msg);

   }

}
```

10º Teste

Criar tratamento com exceção para evitar caracteres inválidos na String

```
/**
*  Inválido - Caracteres especiais não são permitidos, somente
*  letras e  números
*/

@Test(expected=CaracteresEspeciaisNaoPermitido.class)

public void caracteresEspeciaisNaoPermitido() {

   String camelCase = "nome#Composto";

   Conversor.converterCamelCase(camelCase);

}
```

Rotina responsável por identificar caracteres especiais dentro da String

```
private boolean caracteresEspeciaisNaoPermitidos(String original) {

   Pattern p = Pattern.compile("[^a-z0-9 ]", Pattern.CASE_INSENSITIVE);
   Matcher m = p.matcher(original);
   return m.find();

}
```

```
package tddcamelcase;

public class CaracteresEspeciaisNaoPermitido extends RuntimeException {

       public CaracteresEspeciaisNaoPermitido(String msg) {
              super(msg);

       }

}
```

```
public static List<String> converterCamelCase(String original){

   Conversor converter = new Conversor();
  if (converter.comecaComNumero(original)){
      throw new  ComecaComNumerosException("A palavra possui número no seu
      início");

      } else if (converter.caracteresEspeciaisNaoPermitidos(original)) {
            throw new CaracteresEspeciaisNaoPermitido("A palavra possu
      caracteres     inválidos");

      }else{

            listaPalavras = Conversor.desfazerUmCamelCase(original);
         }

         return listaPalavras;

      }
```