

# **INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS**



**RELATÓRIO CAP-389: PROJETO ÁGIL DE SOFTWARE**

**PROFESSOR: DR. EDUARDO GUERRA**

**CAMEL CASE**

**LUIZ WAGNER TAVARES NASCIMENTO**

**SÃO JOSÉ DOS CAMPOS**

**2016**

## 1. OBJETIVO

Apresentar o processo de TDD – Test Driven Development realizado durante o desenvolvimento do Exercício 01 – Camel Case.

## 2. DESENVOLVIMENTO

Primeiramente foi criado um novo projeto Java no Eclipse, feito o import da biblioteca de testes unitário JUnit4 e criada a classe de testes que guiou todo a implementação do método para transformação da cadeia de caracteres em Camel Case.

### 2.1. CLASSE DE TESTE INICIAL

A partir da estrutura de código abaixo, o compilador identificou que não existia a classe `CamelCase`.

- Código de teste (source folder “test”)

```
package br.inpe.cap.projetoagil;

import static org.junit.Assert.assertEquals;
import java.util.List;
import org.junit.Test;

public class TestCamelCase {

    @Test
    public void palavraUnica() {
        List<String> palavras = CamelCase.converterCamelCase("nome");
        String resultado = palavras.get(0);
        assertEquals("nome", resultado);
    }

}
```

Utilizando o wizard do Eclipse, foi solicitado para criar a classe `CamelCase` no source-folder “src”. Em seguida, o compilador indicou que o método `converterCamelCase` não existia e, utilizando novamente o wizard, foi solicitada a criação do referido método.

- Classe de domínio (source folder “src”) – *Antes*

```
package br.inpe.cap.projetoagil;

import java.util.List;

public class CamelCase {

    public static List<String> converterCamelCase(String original) {
        return null;
    }

}
```

A execução do teste falhou, causando `NullPointerException`. Foi criada então uma nova lista para o retorno e o seguinte código foi o suficiente para o primeiro testes passar:

- Classe de domínio (source folder “src”) – *Depois*

```
package br.inpe.cap.projetoagil;

import java.util.ArrayList;
import java.util.List;

public class CamelCase {

    public static List<String> converterCamelCase(String original) {
        List<String> listaStrings = new ArrayList<>();
        listaStrings.add(original);
        return listaStrings ;
    }

}
```

## 2.2. TESTE – PRIMEIRA LETRA MAIÚSCULA

Para fazer passar o teste de uma string com a primeira letra maiúscula, foi apenas criada uma variável para a string convertida e atribuído o valor do parâmetro `original` passando para `toLowerCase`:

- Código de teste (source folder “test”)

```
@Test
public void primeiraLetraMaiuscula() {
    List<String> palavras = CamelCase.converterCamelCase("Nome");
    String resultado = palavras.get(0);
    assertEquals("nome", resultado);
}
```

- Classe de domínio (source folder “src”) – *Antes*

```
package br.inpe.cap.projetoagil;

import java.util.List;

public class CamelCase {

    public static List<String> converterCamelCase(String original) {
        List<String> listaStrings = new ArrayList<>();
        listaStrings.add(original);
        return listaStrings ;
    }

}
```

- Classe de domínio (source folder “src”) – *Depois*

```
package br.inpe.cap.projetoagil;

import java.util.ArrayList;
import java.util.List;

public class CamelCase {

    public static List<String> converterCamelCase(String original) {
        List<String> listaStrings = new ArrayList<>();
        String convertida = original.toLowerCase();
        listaStrings.add(convertida);
        return listaStrings ;
    }

}
```

## 2.3. TESTE – NOME COMPOSTO (PRIMEIRA MINÚSCULA)

No teste com um nome composto já foi necessário utilizar o método `String.split()`, que separa a string original em um array de strings baseado em uma expressão regular (regex). Porém, ao passar a regex para letra maiúscula, o método `split` “some” com o caractere encontrado. Dessa forma, foi substituída a letra maiúscula por um caractere sublinhado (“\_”) + respectiva letra maiúscula, e só então chamado o método `split`, e assim ele separa a string original desconsiderando o caractere sublinhado.

- Código de teste (source folder “test”)

```
@Test
public void nomeCompostoPrimeiraLetraMinuscula() {
    List<String> palavras = CamelCase.converterCamelCase("nomeComposto");
    String resultado1 = palavras.get(0);
    String resultado2 = palavras.get(1);
    assertEquals("nome", resultado1);
    assertEquals("composto", resultado2);
}
```

- Classe de domínio (source folder “src”) – *Antes*

```
public static List<String> converterCamelCase(String original) {
    List<String> listaStrings = new ArrayList<>();
    String convertida = original.toLowerCase();
    listaStrings.add(convertida);
    return listaStrings ;
}
```

- Classe de domínio (source folder “src”) – *Depois*

```
package br.inpe.cap.projetoagil;

import java.util.ArrayList;
import java.util.List;

public class CamelCase {

    private static final String REGEX = "([_A-Z])([A-Z])";
    private static final String SUBLINHADO = "_";
    private static final String REGEX_SUBLINHADO = "$1" + SUBLINHADO + "$2";

    public static List<String> converterCamelCase(String original) {
        List<String> listaStrings = new ArrayList<>();
        String sublinhado = original.replaceAll(REGEX, REGEX_SUBLINHADO);
        String[] separadas = sublinhado.split(SUBLINHADO);
        for (String parte : separadas) {
            parte = parte.toLowerCase();
            listaStrings.add(parte);
        }
        return listaStrings ;
    }
}
```

## 2.4. TESTE – NOME COMPOSTO (PRIMEIRA MAIÚSCULA)

Para o teste também com um nome compostos, porém com a primeira letra maiúscula, o teste passou sem que fosse necessário alterar nada no código do método de conversão.

- Código de teste (source folder “test”)

```
@Test
public void nomeCompostoPrimeiraLetraMaiuscula() {
    List<String> palavras = CamelCase.converterCamelCase("NomeComposto");
    String resultado1 = palavras.get(0);
    String resultado2 = palavras.get(1);
    assertEquals("nome", resultado1);
    assertEquals("composto", resultado2);
}
```

- Classe de domínio (source folder “src”) – *Sem alteração!*

```
package br.inpe.cap.projetoagil;

import java.util.ArrayList;
import java.util.List;

public class CamelCase {

    private static final String REGEX = "([^_A-Z])([A-Z])";
    private static final String SUBLINHADO = "_";
    private static final String REGEX_SUBLINHADO = "$1" + SUBLINHADO + "$2";

    public static List<String> converterCamelCase(String original) {
        List<String> listaStrings = new ArrayList<>();
        String sublinhado = original.replaceAll(REGEX, REGEX_SUBLINHADO);
        String[] separadas = sublinhado.split(SUBLINHADO);
        for (String parte : separadas) {
            parte = parte.toLowerCase();
            listaStrings.add(parte);
        }
        return listaStrings ;
    }
}
```

## 2.5. TESTE – NOME COMPOSTO (SEPARADO POR SUBLINHADO)

Para testar o caso de um nome composto separado por sublinhado (foge da especificação CamelCase, porém utiliza o mesmo caractere especial escolhido para o método `split`) foi criado o teste abaixo. Ainda assim não foi necessário alterar nada no método de conversão.

- Código de teste (source folder “test”)

```
@Test
public void nomeCompostoSeparadoPorSublinhado() {
    List<String> palavras =
        CamelCase.converterCamelCase("Nome_Separado_Por_Sublinhado");
    String resultado1 = palavras.get(0);
    String resultado2 = palavras.get(1);
    String resultado3 = palavras.get(2);
    String resultado4 = palavras.get(3);
    assertEquals("nome", resultado1);
    assertEquals("separado", resultado2);
    assertEquals("por", resultado3);
    assertEquals("sublinhado", resultado4);
}
```

- Classe de domínio (source folder “src”) – *Sem alteração!*

```
package br.inpe.cap.projetoagil;

import java.util.ArrayList;
import java.util.List;

public class CamelCase {

    private static final String REGEX = "([_A-Z])([A-Z])";
    private static final String SUBLINHADO = "_";
    private static final String REGEX_SUBLINHADO = "$1" + SUBLINHADO + "$2";

    public static List<String> converterCamelCase(String original) {
        List<String> listaStrings = new ArrayList<>();
        String sublinhado = original.replaceAll(REGEX, REGEX_SUBLINHADO);
        String[] separadas = sublinhado.split(SUBLINHADO);
        for (String parte : separadas) {
            parte = parte.toLowerCase();
            listaStrings.add(parte);
        }
        return listaStrings ;
    }
}
```

## 2.6. TESTE – SIGLA

No teste em quem a palavra é apenas uma sigla, ela não precisa ser passada para *lowerCase*. Desta forma, foi acrescentada uma condição com o método privado `isSigla`, durante a transformação das partes separadas pelo método `split`.

- Código de teste (source folder “test”)

```
@Test
public void sigla() {
    List<String> palavras = CamelCase.converterCamelCase("CPF");
    String resultado1 = palavras.get(0);
    assertEquals("CPF", resultado1);
}
```

- Classe de domínio (source folder “src”) – *Antes*

```
public class CamelCase {

    private static final String REGEX = "([_A-Z])([A-Z])";
    private static final String SUBLINHADO = "_";
    private static final String REGEX_SUBLINHADO = "$1" + SUBLINHADO + "$2";

    public static List<String> converterCamelCase(String original) {
        List<String> listaStrings = new ArrayList<>();
        String sublinhado = original.replaceAll(REGEX, REGEX_SUBLINHADO);
        String[] separadas = sublinhado.split(SUBLINHADO);
        for (String parte : separadas) {
            parte = parte.toLowerCase();
            listaStrings.add(parte);
        }
        return listaStrings ;
    }
}
```

- Classe de domínio (source folder “src”) – *Depois*

```
package br.inpe.cap.projetoagil;

import java.util.ArrayList;
import java.util.List;

public class CamelCase {

    private static final String REGEX = "([_A-Z])([A-Z])";
    private static final String SUBLINHADO = "_";
    private static final String REGEX_SUBLINHADO = "$1" + SUBLINHADO + "$2";

    public static List<String> converterCamelCase(String original) {
        List<String> listaStrings = new ArrayList<>();
        String sublinhado = original.replaceAll(REGEX, REGEX_SUBLINHADO);
        String[] separadas = sublinhado.split(SUBLINHADO);
        for (String parte : separadas) {
            if(!isSigla(parte)) {
                parte = parte.toLowerCase();
            }
            listaStrings.add(parte);
        }

        return listaStrings ;
    }

    private static boolean isSigla(String parte) {
        return parte.equals(parte.toUpperCase());
    }
}
```

## 2.7. TESTE – PALAVRA SEGUIDA DE SIGLA

Foi incluído o teste para uma palavra seguida de uma sigla. Não foi necessário alterar nada no código.

- Código de teste (source folder “test”)

```
@Test
public void palavraESigla() {
    List<String> palavras = CamelCase.converterCamelCase("númeroCPF");
    String resultado1 = palavras.get(0);
    String resultado2 = palavras.get(1);
    assertEquals("número", resultado1);
    assertEquals("CPF", resultado2);
}
```

- Classe de domínio (source folder “src”) – *Sem alteração!*

```
package br.inpe.cap.projetoagil;

import java.util.ArrayList;
import java.util.List;

public class CamelCase {

    private static final String REGEX = "([^_A-Z])([A-Z])";
    private static final String SUBLINHADO = "_";
    private static final String REGEX_SUBLINHADO = "$1" + SUBLINHADO + "$2";

    public static List<String> converterCamelCase(String original) {
        List<String> listaStrings = new ArrayList<>();
        String sublinhado = original.replaceAll(REGEX, REGEX_SUBLINHADO);
        String[] separadas = sublinhado.split(SUBLINHADO);
        for (String parte : separadas) {
            if(!isSigla(parte)) {
                parte = parte.toLowerCase();
            }
            listaStrings.add(parte);
        }
        return listaStrings ;
    }

    private static boolean isSigla(String parte) {
        return parte.equals(parte.toUpperCase());
    }
}
```



## 2.8. TESTE – ADICIONANDO PALAVRA APÓS A SIGLA

Quando foi adicionada uma palavra após a sigla, o teste não passou. Foi então necessário tratar a separação da próxima palavra começada com letra maiúscula, adicionando então uma condicional na *regex*.

- Código de teste (source folder “test”)

```
@Test
public void adicionadaPalavraAposASigla() {
    List<String> palavras = CamelCase.converterCamelCase("númeroCPFContribuinte");
    String resultado1 = palavras.get(0);
    String resultado2 = palavras.get(1);
    String resultado3 = palavras.get(2);
    assertEquals("número", resultado1);
    assertEquals("CPF", resultado2);
    assertEquals("contribuinte", resultado3);
}
```

- Classe de domínio (source folder “src”) – Antes

```
public class CamelCase {

    private static final String REGEX = "([_A-Z])([A-Z])";
    private static final String SUBLINHADO = "_";
    private static final String REGEX_SUBLINHADO = "$1" + SUBLINHADO + "$2";

    public static List<String> converterCamelCase(String original) {
        List<String> listaStrings = new ArrayList<>();
        String sublinhado = original.replaceAll(REGEX, REGEX_SUBLINHADO);
        String[] separadas = sublinhado.split(SUBLINHADO);
        for (String parte : separadas) {
            if(!isSigla(parte)) {
                parte = parte.toLowerCase();
            }
            listaStrings.add(parte);
        }
        return listaStrings ;
    }

    private static boolean isSigla(String parte) {
        return parte.equals(parte.toUpperCase());
    }
}
```

- Classe de domínio (source folder “src”) – Depois

```
public class CamelCase {

    private static final String REGEX = "([_A-Z])([A-Z])|(?<=[A-Z])(?=[A-Z][a-z]) ";
    private static final String SUBLINHADO = "_";
    private static final String REGEX_SUBLINHADO = "$1" + SUBLINHADO + "$2";

    public static List<String> converterCamelCase(String original) {
        List<String> listaStrings = new ArrayList<>();
        String sublinhado = original.replaceAll(REGEX, REGEX_SUBLINHADO);
        String[] separadas = sublinhado.split(SUBLINHADO);
        for (String parte : separadas) {
            if(!isSigla(parte)) {
                parte = parte.toLowerCase();
            }
            listaStrings.add(parte);
        }
        return listaStrings ;
    }

    private static boolean isSigla(String parte) {
        return parte.equals(parte.toUpperCase());
    }
}
```

## 2.9. TESTE – PALAVRAS COM NÚMERO INTERCALADO

O teste de palavras com um número intercalado também não passou sem alteração. A *regex* foi alterada novamente, fazendo o tratamento agora dos números intercalados.

- Código de teste (source folder “test”)

```
@Test
public void palavrasComNumeroIntercalado() {
    List<String> palavras = CamelCase.converterCamelCase("recupera10Primeiros");
    String resultado1 = palavras.get(0);
    String resultado2 = palavras.get(1);
    String resultado3 = palavras.get(2);
    assertEquals("recupera", resultado1);
    assertEquals("10", resultado2);
    assertEquals("primeiros", resultado3);
}
```

- Classe de domínio (source folder “src”) – Antes

```
public class CamelCase {

    private static final String REGEX = "([_A-Z])([A-Z])|(?<=[A-Z])(?=[A-Z][a-z]) ";
    private static final String SUBLINHADO = "_";
    private static final String REGEX_SUBLINHADO = "$1" + SUBLINHADO + "$2";

    public static List<String> converterCamelCase(String original) {
        List<String> listaStrings = new ArrayList<>();
        String sublinhado = original.replaceAll(REGEX, REGEX_SUBLINHADO);
        String[] separadas = sublinhado.split(SUBLINHADO);
        for (String parte : separadas) {
            if(!isSigla(parte)) {
                parte = parte.toLowerCase();
            }
            listaStrings.add(parte);
        }
        return listaStrings ;
    }
    private static boolean isSigla(String parte) {
        return parte.equals(parte.toUpperCase());
    }
}
```

- Classe de domínio (source folder “src”) – Depois

```
public class CamelCase {

    private static final String REGEX =
        "([_A-Z])([A-Z])|(?<=[A-Z])(?=[A-Z][a-z])|(?<=[a-z])(?=[0-9])";
    private static final String SUBLINHADO = "_";
    private static final String REGEX_SUBLINHADO = "$1" + SUBLINHADO + "$2";

    public static List<String> converterCamelCase(String original) {
        List<String> listaStrings = new ArrayList<>();
        String sublinhado = original.replaceAll(REGEX, REGEX_SUBLINHADO);
        String[] separadas = sublinhado.split(SUBLINHADO);
        for (String parte : separadas) {
            if(!isSigla(parte)) {
                parte = parte.toLowerCase();
            }
            listaStrings.add(parte);
        }
        return listaStrings ;
    }
    private static boolean isSigla(String parte) {
        return parte.equals(parte.toUpperCase());
    }
}
```

## 2.10. TESTE – APENAS CONECTOR E

No teste da palavra ser apenas a preposição “e”, ela não deve ser tratada como uma sigla (o mesmo aconteceria com o artigo/preposição “a”). Então nesse caso foi adicionada uma condição no método privado `isSigla()` para que considere apenas palavras com 2 ou mais letras.

- Código de teste (source folder “test”)

```
@Test
public void apenasLetraE() {
    List<String> palavras = CamelCase.converterCamelCase("E");
    String resultado1 = palavras.get(0);
    assertEquals("e", resultado1);
}
```

- Classe de domínio (source folder “src”) – *Antes*

```
public class CamelCase {

    private static final String REGEX =
        "([^_A-Z])([A-Z])|(?<=[A-Z])(?=[A-Z][a-z])|(?<=[a-z])(?=[0-9])";
    private static final String SUBLINHADO = "_";
    private static final String REGEX_SUBLINHADO = "$1" + SUBLINHADO + "$2";

    public static List<String> converterCamelCase(String original) {
        List<String> listaStrings = new ArrayList<>();
        String sublinhado = original.replaceAll(REGEX, REGEX_SUBLINHADO);
        String[] separadas = sublinhado.split(SUBLINHADO);
        for (String parte : separadas) {
            if(!isSigla(parte)) {
                parte = parte.toLowerCase();
            }
            listaStrings.add(parte);
        }
        return listaStrings ;
    }
    private static boolean isSigla(String parte) {
        return parte.equals(parte.toUpperCase());
    }
}
```

- Classe de domínio (source folder “src”) – *Depois*

```
public class CamelCase {

    private static final String REGEX =
        "([^_A-Z])([A-Z])|(?<=[A-Z])(?=[A-Z][a-z])|(?<=[a-z])(?=[0-9])";
    private static final String SUBLINHADO = "_";
    private static final String REGEX_SUBLINHADO = "$1" + SUBLINHADO + "$2";

    public static List<String> converterCamelCase(String original) {
        List<String> listaStrings = new ArrayList<>();
        String sublinhado = original.replaceAll(REGEX, REGEX_SUBLINHADO);
        String[] separadas = sublinhado.split(SUBLINHADO);
        for (String parte : separadas) {
            if(!isSigla(parte)) {
                parte = parte.toLowerCase();
            }
            listaStrings.add(parte);
        }
        return listaStrings ;
    }
    private static boolean isSigla(String parte) {
        return parte.length() > 1 && parte.equals(parte.toUpperCase());
    }
}
```

## 2.11. TESTE – PALAVRA INVÁLIDA COMEÇANDO COM NÚMERO

Ao iniciar os testes com palavras inválidas, foi utilizado o parâmetro `expected` da anotação `@Test`. A execução do teste falhou devido a exceções diferentes esperadas, uma vez que o método de teste falharia apenas por chamar o método de conversão passando uma palavra inválida. A expressão regular foi alterada parada para `lowerCase`. Desta forma, foi acrescentada uma condição na transformação das partes separadas pelo método `split`.

- Código de teste (source folder “test”)

```
@Test(expected=IllegalArgumentException.class)
public void palavraInvalidaComecandoComNumero() {
    CamelCase.converterCamelCase("10Primeiros");
    fail("Palavra não deveria começar com números.");
}
```

- Classe de domínio (source folder “src”) – Antes

```
public class CamelCase {

    private static final String REGEX =
        "([^_A-Z])([A-Z])|(?<=[A-Z])(?=[A-Z][a-z])|(?<=[a-z])(?=[0-9])";
    private static final String SUBLINHADO = "_";
    private static final String REGEX_SUBLINHADO = "$1" + SUBLINHADO + "$2";

    public static List<String> converterCamelCase(String original) {
        List<String> listaStrings = new ArrayList<>();
        String sublinhado = original.replaceAll(REGEX, REGEX_SUBLINHADO);
        String[] separadas = sublinhado.split(SUBLINHADO);
        for (String parte : separadas) {
            if(!isSigla(parte)) {
                parte = parte.toLowerCase();
            }
            listaStrings.add(parte);
        }
        return listaStrings ;
    }

    private static boolean isSigla(String parte) {
        return parte.length() > 1 && parte.equals(parte.toUpperCase());
    }

}
```

- Classe de domínio (source folder “src”) – Depois

```
public class CamelCase {

    private static final String REGEX =
        "([^_A-Z])([A-Z])|(?<=[A-Z])(?=[A-Z][a-z])|(?<=[a-z])(?=[0-9])";
    private static final String SUBLINHADO = "_";
    private static final String REGEX_SUBLINHADO = "$1" + SUBLINHADO + "$2";

    private static final Pattern PATTERN_NUMEROS = Pattern.compile("^(\d+.)");

    public static List<String> converterCamelCase(String original) {
        checkValidString(original);
        List<String> listaStrings = new ArrayList<>();
        String sublinhado = original.replaceAll(REGEX, REGEX_SUBLINHADO);
        String[] separadas = sublinhado.split(SUBLINHADO);
        for (String parte : separadas) {
            if(!isSigla(parte)) {
                parte = parte.toLowerCase();
            }
            listaStrings.add(parte);
        }
        return listaStrings ;
    }

    private static boolean isSigla(String parte) {
        return parte.length() > 1 && parte.equals(parte.toUpperCase());
    }

}
```

```

    }

    private static void checkValidString(String original) {
        if (PATTERN_NUMEROS.matcher(original).matches()) {
            throw new IllegalArgumentException("Palavra não deve começar com números.");
        }
    }
}

```

## 2.12. TESTE – PALAVRA INVÁLIDA CONTENDO CARACTERE ESPECIAL

O teste de uma palavra inválida contendo caracteres especiais não obteve sucesso até o momento da escrita deste relatório, sendo que diversas expressões regulares foram testadas para validar caracteres não-alfanuméricos, porém todos sem sucesso:

- `^[a-zA-Z0-9]*$`
- `^[A-Za-z0-9]`
- `[$&+, : ; = ? @ # | ' < > . ^ * ( ) % ! -]`
- `^[\\w\\s]`
- `^[\\p{L}\\d\\s_]`
- `^ \\p{Alnum}`
- `\\p{Punct}`
- `^[\\W]`

- Código de teste (source folder “test”)

```

@Test(expected=IllegalArgumentException.class)
public void palavraInvalidaComecandoComNumero() {
    CamelCase.converterCamelCase("nome#Composto");
    fail("Caracteres especiais não deveriam ser permitidos.");
}

```

- Classe de domínio (source folder “src”) – *Antes*

```

public class CamelCase {

    private static final String REGEX =
        "([^_A-Z])([A-Z])|(?<=[A-Z])(?=[A-Z][a-z])|(?<=[a-z])(?=[0-9])";
    private static final String SUBLINHADO = "_";
    private static final String REGEX_SUBLINHADO = "$1" + SUBLINHADO + "$2";

    private static final Pattern PATTERN_NUMEROS = Pattern.compile("^(\\d+\\.*)");

    public static List<String> converterCamelCase(String original) {
        checkValidString(original);
        List<String> listaStrings = new ArrayList<>();
        String sublinhado = original.replaceAll(REGEX, REGEX_SUBLINHADO);
        String[] separadas = sublinhado.split(SUBLINHADO);
        for (String parte : separadas) {
            if (!isSigla(parte)) {
                parte = parte.toLowerCase();
            }
            listaStrings.add(parte);
        }
        return listaStrings ;
    }

    private static boolean isSigla(String parte) {
        return parte.length() > 1 && parte.equals(parte.toUpperCase());
    }

    private static void checkValidString(String original) {
        if (PATTERN_NUMEROS.matcher(original).matches()) {
            throw new IllegalArgumentException("Palavra não deve começar com números.");
        }
    }
}

```

```
}  
}
```

- Classe de domínio (source folder “src”) – *Depois*

```
public class CamelCase {  
  
    private static final String REGEX =  
        "([_A-Z])([A-Z])|(?<=[A-Z])(?=[A-Z][a-z])|(?<=[a-z])(?=[0-9])";  
    private static final String SUBLINHADO = "_";  
    private static final String REGEX_SUBLINHADO = "$1" + SUBLINHADO + "$2";  
  
    private static final Pattern PATTERN_NUMEROS = Pattern.compile("^\\d+.*");  
    private static final Pattern PATTERN_CARACTERES_ESPECIAIS = Pattern.compile("(^\\W$)");  
  
    public static List<String> converterCamelCase(String original) {  
        checkValidString(original);  
        List<String> listaStrings = new ArrayList<>();  
        String sublinhado = original.replaceAll(REGEX, REGEX_SUBLINHADO);  
        String[] separadas = sublinhado.split(SUBLINHADO);  
        for (String parte : separadas) {  
            if (!isSigla(parte)) {  
                parte = parte.toLowerCase();  
            }  
            listaStrings.add(parte);  
        }  
        return listaStrings ;  
    }  
  
    private static boolean isSigla(String parte) {  
        return parte.length() > 1 && parte.equals(parte.toUpperCase());  
    }  
  
    private static void checkValidString(String original) {  
        if (PATTERN_NUMEROS.matcher(original).matches()) {  
            throw new IllegalArgumentException("Palavra não deve começar com números.");  
        }  
        if (PATTERN_CARACTERES_ESPECIAIS.matcher(original).matches()) {  
            throw new IllegalArgumentException("Caracteres especiais não são permitidos,  
somente letras e números.");  
        }  
    }  
}
```