Relatório – Quebra de Strings com CamelCase

ITA - TDD – Desenvolvimento de Software Guiado por Testes (Coursera)

01 - Teste para string vazia:

```
@Test
public void testaStringVazia() {
   ConvertCamelCase c = new ConvertCamelCase();
   List<String> lista = c.converterCamelCase("").
   assertTrue(lista.isEmpty());
}
```

Com o teste criado e falhado, criamos a classe ConvertCamelCase.java com o seguinte trecho de código:

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

public class ConvertCamelCase {

   public List<String> converterCamelCase(String string) {
      List<String> lista = new ArrayList<String>();
      return lista;
   }

}
```

02 – Teste ok. O próximo teste será para verificar string minúscula:

```
@Test
public void testaStringMinuscula() {
   ConvertCamelCase c = new ConvertCamelCase();
   List<String> listaRecebida = c.converterCamelCase("nome");
   List<String> listaEsperada = new ArrayList<String>();
   listaEsperada.add("nome");
   assertEquals(listaRecebida, listaEsperada);
}
```

Teste criado e falhado, ajustamos o trecho de código no ConvertCamelCase.java conforme abaixo:

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

public class ConvertCamelCase {
   public List<String> converterCamelCase(String string) {
      List<String> lista = new ArrayList<String>();
      lista.add(string);
      return lista;
   }
}
```

O segundo teste passa mas o primeiro teste dá erro. Ocorre que o primeiro teste espera uma lista com string vazia e não uma lista vazia. Ajustamos o código do primeiro teste, conforme abaixo, e ambos os testes passam.

```
@Test
public void testaStringVazia() {
   ConvertCamelCase c = new ConvertCamelCase();
   List<String> listaRecebida = c.converterCamelCase("");
   List<String> listaEsperada = new ArrayList<String>();
   listaEsperada.add("");
   assertEquals(listaRecebida, listaEsperada);
}
```

Agora vamos refatorar a classe de testes separando as declarações de variáveis e inicializações

```
private List<String> listaEsperada;
private ConvertCamelCase c;
private List<String> listaRecebida;

@BeforeEach
public void inicializador() {
   c = new ConvertCamelCase();
   listaEsperada = new ArrayList<String>();
}
```

03 – Agora adicionamos um teste para uma única palavra:

```
@Test
public void testaUnicaPalavra() {
   listaRecebida = c.converterCamelCase("Nome");
   listaEsperada.add("nome");
   assertEquals(listaRecebida, listaEsperada);
}
```

O teste retorna “Nome” quando deveria retornar “nome”. Para corrigir essa falha realizamos as seguintes alterações:

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

public class ConvertCamelCase {
   public List<String> converterCamelCase(String string) {
      List<String> lista = new ArrayList<String>();
      lista.add(string.toLowerCase());
      return lista;
   }
}
```

04 – Agora iremos testar palavras compostas:

```
@Test
public void testaPalavraComposta() {
   listaRecebida = c.converterCamelCase("nomeComposto");
   listaEsperada.add("nome");
   listaEsperada.add("composto");
   assertEquals(listaRecebida, listaEsperada);
}
```

O teste apresentou falha, agora iremos ajustar o código para passar no teste:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class ConvertCamelCase {
   public List<String> converterCamelCase(String string) {
      List<String> lista = new ArrayList<String>();
      lista = Arrays.asList(string.split("[A-Z]"));
      return lista;
   }
}
```

Com a alteração, além do testaPalavraComposta, falhou também testaUnicaPalavra, pois o split está eliminando os caracteres maiúsculos. Abaixo ajustamos a string de regex utilizada no split para "(?=[A-Z])", para corrigir o problema:

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

public class ConvertCamelCase {
   public List<String> converterCamelCase(String string) {
      String[] listaComMaiusculas = string.split("(?=[A-Z])");
      List<String> lista = new ArrayList<String>();
      for(String palavra: listaComMaiusculas) {
         lista.add(palavra.toLowerCase());
      }
      return lista;
   }
}
```

Todos os testes passam. Agora vamos realizar refatoramento da estética do código:

```
public List<String> converterCamelCase(String string) {
   String[] listaComMaiusculas = string.split("(?=[A-Z])");
   List<String> lista = new ArrayList<String>();
   for(String palavra: listaComMaiusculas)
      lista.add(palavra.toLowerCase());
   return lista;
}
```

05 – Para o próximo teste, iremos verificar palavras compostas maiúsculas:

```
@Test
public void testaPalavraCompostaMaiuscula() {
   listaRecebida = c.converterCamelCase("NomeComposto");
   listaEsperada.add("nome");
   listaEsperada.add("composto");
   assertEquals(listaRecebida, listaEsperada);
}
```

06 – O teste passa direto, o próximo teste é para verificar palavras inteiras maiúsculas:

```
@Test
public void testaPalavraInteiraMaiuscula() {
   listaRecebida = c.converterCamelCase("CPF");
   listaEsperada.add("CPF");
   assertEquals(listaEsperada, listaRecebida);
}
```

O teste falha, o método retorna (“c” , “p”, “f”). Realizamos os ajustes abaixo na busca da solução ideal:

```
import java.util.ArrayList;
import java.util.List;
import java.util.regex.*;

public class ConvertCamelCase {
   public List<String> converterCamelCase(String string) {
      List<String> lista = new ArrayList<String>();
      if(string.toUpperCase().equals(string)) {
         lista.add(string);
      } else {
         String[] listaComMaiusculas = string.split("(?=([A-Z]+))");
         for(String palavra: listaComMaiusculas)
            lista.add(palavra.toLowerCase());
      }
      return lista;
   }
}
```

Teste ok, próximo teste..

07 – O teste a seguir verifica palavras inteiras maiúsculas compostas:

```
@Test
public void testaPalavraInteiraMaisuculaComposta() {
   listaRecebida = c.converterCamelCase("numeroCPF");
   listaEsperada.add("numero");
   listaEsperada.add("CPF");
   assertEquals(listaEsperada, listaRecebida);
}
```

O teste falha, e realizamos os ajustes a seguir:

```
import java.util.ArrayList;
import java.util.List;
import java.util.regex.*;

public class ConvertCamelCase {
   public List<String> converterCamelCase(String string) {
      List<String> lista = new ArrayList<String>();
      Pattern pattern = Pattern.compile("([A-Z][a-z]+)|([a-z]+)|([A-Z]+)");
      Matcher matcher = pattern.matcher(string);
      while(matcher.find()) {
         lista.add(matcher.group(0));
      }
      return lista;
   }
}
```

Foi substituido o método 'split(String regex)' da classe String para dividir a string automaticamente, utilizamos as classes Pattern e Matcher, do pacote básico de expressões regulares da linguagem Java, para selecionar os grupos de interesse, No caso, a primeira parte do padrão "([A-Z][a-z]+)" padrão seleciona sequências começadas por uma letra maiúscula seguida por uma cadeia de letras maiúsculas. A segunda parte "([a-z]+)" seleciona uma cadeia de letras minúsculas qualquer, e a terceira parte "([A-Z]+)" faz o mesmo para cadeias de maiúsculas. Vale notar que o padrão está em ordem de prioridade.

Após a alteração vários testes começaram a falhar, as palavras que começavam com letra maiúscula não estão mais sendo retornadas com letra minúscula. Com as alterações abaixo corrigimos o problema:

```
import java.util.ArrayList;
import java.util.List;
import java.util.regex.*;

public class ConvertCamelCase {
   public List<String> converterCamelCase(String string) {
      List<String> lista = new ArrayList<String>();
      if(string.isEmpty()) {
         lista.add("");
      }
      Pattern pattern = Pattern.compile("([A-Z][a-z]+)|([a-z]+)|([A-Z]+)");
      Matcher matcher = pattern.matcher(string);
      while(matcher.find()) {
         if(!matcher.group(0).toUpperCase().equals(matcher.group(0)))
            lista.add(matcher.group().toLowerCase());
         else
            lista.add(matcher.group(0));
         }
         return lista;
   }
}
```

Os testes passam, a seguir efetuamos refatoração:

```
public class ConvertCamelCase {
   List<String> listaResposta;

   public List<String> converterCamelCase(String string) {
      listaResposta = new ArrayList<String>();
      if(string.isEmpty())
         listaResposta.add("");
      Pattern pattern = Pattern.compile("([A-Z][a-z]+)|([a-z]+)|([A-Z]+)");
      Matcher matcher = pattern.matcher(string);
      while(matcher.find())
         listaResposta.add(formataResposta(matcher.group(0)));
      return listaResposta;
   }

   private String formataResposta(String string) {
      if(!inteiraMaiuscula(string)) {
         return string.toLowerCase();
      } else {
         return string;
      }
   }

   private boolean inteiraMaiuscula(String palavra) {
      return palavra.toUpperCase().equals(palavra);
   }
}
```

08 – Agora vamos passar para o teste das palavras inteiras maiúsculas entre palavras:

```
@Test
public void testaPalavraInteiraMaiusculaEntrePalavras() {
   listaRecebida = c.converterCamelCase("numeroCPFContribuinte");
   listaEsperada.add("numero");
   listaEsperada.add("CPF");
   listaEsperada.add("contribuinte");
   assertEquals(listaEsperada, listaRecebida);
}
```

O teste falha pois o "C" de contribuinte entra no mesmo grupo de "CPF" ("numero", "CPFC", "ontribuinte"). Corrigimos o problema substituindo o último grupo do padrão de regex utilizado por [A-Z]+(?![a-z]+), utilizando um negative lookahead para fazer com que cadeias maiúsculas não selecionem caracteres seguidos por minúsculas.

09 – O próximo será para testarmos números entre palavras:

```
@Test
public void testaNumeroEntrePalavras() {
   listaRecebida = c.converterCamelCase("recupera10Primeiros");
   listaEsperada.add("recupera");
   listaEsperada.add("10");
   listaEsperada.add("primeiros");
   assertEquals(listaEsperada, listaRecebida);
}
```

O teste falha, o regex atual não pega números. Realizamos os ajustes acrescentando "|([0-9]+)" ao padrão atual.

10 – Para o próximo teste iremos lançar uma exceção de CamelCase inválido quando a string passada é iniciada com um número:

```
@Test
void testaPalavraComecandoPorNumero() {
    assertThrows(CamelCaseInvalidoException.class, () -> {
        listaRecebida = c.converterCamelCase("10Primeiros");
    });
}
```

Criamos então a classe CamelCaseInvalidoException e fazemos com que ela seja lançada no caso descrito acima através das seguintes alterações no método principal:

```
public List<String> converterCamelCase(String string) {
   listaResposta = new ArrayList<String>();
   if(string.isEmpty()) {
      listaResposta.add("");
   } else {
      char[] firstCharacter = new char[1];
      string.getChars(0, 1, firstCharacter, 0);
      if(firstCharacter[0] > '0' && firstCharacter[0] < '9')
         throw new CamelCaseInvalidoException("Nao pode comecar com
numeros");
      Pattern pattern =
Pattern.compile("([A-Z][a-z]+)|([a-z]+)|([A-Z]+(?![a-z]+))|([0-9]+)");
      Matcher matcher = pattern.matcher(string);
      while(matcher.find())
         listaResposta.add(formataResposta(matcher.group(0)));
   }
   return listaResposta;
}
```

Realizamos refatoração criando um novo método auxiliar e ajustando o método principal:

```
private boolean comecaComNumero(String string) {
   char[] firstCharacter = new char[1];
   string.getChars(0, 1, firstCharacter, 0);
   return (firstCharacter[0] > '0' && firstCharacter[0] < '9');
}
```

```
public List<String> converterCamelCase(String string) {
   listaResposta = new ArrayList<String>();
   if(string.isEmpty()) {
      listaResposta.add("");
   } else if (comecaComNumero(string)) {
      throw new CamelCaseInvalidoException("Nao pode comecar com numeros");
   } else {
      Pattern pattern =
Pattern.compile("([A-Z][a-z]+)|([a-z]+)|([A-Z]+(?![a-z]+))|([0-9]+)");
      Matcher matcher = pattern.matcher(string);
      while(matcher.find())
         listaResposta.add(formataResposta(matcher.group(0)));
   }
   return listaResposta;
}
```

11 – Mais um teste, agora vamos testar caracteres especiais:

```
@Test
void testaPalavraComCaracterEspecial() {
    assertThrows(CamelCaseInvalidoException.class, () -> {
        listaRecebida = c.converterCamelCase("nome#Composto");
    });
}
```

O teste deverá lançar a exceção CamelCase inválido caso sejam encontrados caracteres especiais na string recebida. Para isso criaremos o método auxiliar a seguir:

```
private boolean contemCaracteresEspeciais(String string) {
   Pattern pattern = Pattern.compile("[^A-Za-z0-9]");
   Matcher matcher = pattern.matcher(string);
   return matcher.find();
}
```

Esse método é introduzido ao método principal:

```
public List<String> converterCamelCase(String string) {
   listaResposta = new ArrayList<String>();
   if(string.isEmpty()) {
      listaResposta.add("");
   } else if (comecaComNumero(string)) {
      throw new CamelCaseInvalidoException("Nao pode comecar com numeros");
   } else if (contemCaracteresEspeciais(string)) {
      throw new CamelCaseInvalidoException("Nao pode conter caracteres
especiais");
   } else {
      Pattern pattern =
Pattern.compile("([A-Z][a-z]+)|([a-z]+)|([A-Z]+(?![a-z]+))|([0-9]+)");
      Matcher matcher = pattern.matcher(string);
      while(matcher.find())
         listaResposta.add(formataResposta(matcher.group(0)));
   }
   return listaResposta;
}
```

Enfim todos os testes passaram.