**INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS**

****

**RELATÓRIO CAP-389: Projeto ágil de software**

**Professor: DR. eduardo guerra**

**Exercício 2:**

**CAIXA ELETRÔNICO**

**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 02 – Caixa Eletrônico.

1. **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 toda a implementação da simulação das operações de um Caixa Eletrônico.

* 1. **Classe de teste inicial – API DEFINITION**

## A partir da estrutura de código abaixo, o compilador identificou que não existia a classe CaixaEletronico. Este primeiro caso de teste foi do tipo *“API DEFINITION”*, de forma a definir a estrutura da classe e seus métodos iniciais.

* Código de teste (source folder “test”)

**package br.inpe.cap.projetoagil;**

**import static org.junit.Assert.assertEquals;**

**import org.junit.Test;**

**public class TestCaixaEletronico {**

@Test

**public** **void** exibicaoMensagensCaixaEletronico() {

CaixaEletronico caixa = **new** CaixaEletronico();

*assertEquals*("Usuário Autenticado", caixa.logar());

*assertEquals*("Retire seu dinheiro", caixa.sacar());

*assertEquals*("Depósito recebido com sucesso", caixa.depositar());

*assertEquals*("O saldo é R$xx,xx", caixa.saldo());

}

}

## Utilizando o wizard do Ecplise, foi solicitado para criar a classe CaixaEletronico no source-folder “src”.

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

**package** br.inpe.cap.projetoagil;

**public** **class** CaixaEletronico {

**public** String logar() {

**return** **null**;

}

**public** String sacar() {

**return** **null**;

}

**public** String depositar() {

**return** **null**;

}

**public** String saldo() {

**return** **null**;

}

}

## A execução do teste falhou, causando NullPointerException. Foi criada então as Strings de retorno de cada método, resultando na passagem deste primeiro caso de teste:

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

**package** br.inpe.cap.projetoagil;

**public** **class** CaixaEletronico {

**public** String logar() {

**return** "Usuário Autenticado";

}

**public** String sacar() {

**return** "Retire seu dinheiro";

}

**public** String depositar() {

**return** "Depósito recebido com sucesso";

}

**public** String saldo() {

**return** "O saldo é R$xx,xx";

}

}

* 1. **teste – recuperar conta corrente**

## Foi adicionado parâmetro numeroConta para exercitar o ServicoRemoto que deve recuperar a conta corrente que será executada as operações de um usuário naquele CaixaEletronico. Foi criado o teste recuperarContaCorrente e para fazê-lo passar utilizando o Mock do objeto ServicoRemoto, foi criada a interface, a ContaCorrente que seria retornada:

* Código de teste (source folder “test”)

@Rule

**public** JUnitRuleMockery ctx = **new** JUnitRuleMockery();

@Test

**public** **void** recuperarConta() {

CaixaEletronico caixa = **new** CaixaEletronico();

**final** ServicoRemoto servicoRemotoMock = ctx.mock(ServicoRemoto.**class**);

caixa.setServicoRemoto(servicoRemotoMock);

String numeroContaTeste = "123456";

ContaCorrente contaCorrente = **new** ContaCorrente(numeroContaTeste);

ctx.checking(**new** Expectations() {{

oneOf(servicoRemotoMock).recuperarConta(numeroContaTeste);

will(*returnValue*(contaCorrente));

}});

*assertEquals*("Usuário Autenticado", caixa.logar(numeroContaTeste));

}

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

**package** br.inpe.cap.projetoagil;

**public** **class** CaixaEletronico {

**public** String logar() {

**return** "Usuário Autenticado";

}

**public** String sacar() {

**return** "Retire seu dinheiro";

}

**public** String depositar() {

**return** "Depósito recebido com sucesso";

}

**public** String saldo() {

**return** "O saldo é R$xx,xx";

}

}

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

**package** br.inpe.cap.projetoagil;

**private** ServicoRemoto servicoRemoto;

**private** ContaCorrente contaCorrenteAtual;

**public** String logar(String numeroConta) {

**this**.contaCorrenteAtual = servicoRemoto.recuperarConta(numeroConta);

**if**(**this**.contaCorrenteAtual != **null**) {

**return** "Usuário Autenticado";

} **else** {

**return** "Usuário não encontrado";

}

}

## A execução do teste resultou em sucesso, porém o 1º teste passou a não funcionar mais, sendo que agora a classe CaixaEletronico depende do ServicoRemoto para realizar suas operações. Foi então ajustado o 1º teste colocando o mock do ServicoRemoto e o teste voltou a funcionar normalmente:

* Código de teste (source folder “test”) – ***Antes***

@Test

**public** **void** exibicaoMensagensCaixaEletronico() {

CaixaEletronico caixa = **new** CaixaEletronico();

*assertEquals*("Usuário Autenticado", caixa.logar());

*assertEquals*("Retire seu dinheiro", caixa.sacar());

*assertEquals*("Depósito recebido com sucesso", caixa.depositar());

*assertEquals*("O saldo é R$xx,xx", caixa.saldo());

}

* Código de teste (source folder “test”) – ***Depois***

@Rule

**public** JUnitRuleMockery ctx = **new** JUnitRuleMockery();

@Test

**public** **void** exibicaoMensagensCaixaEletronico() {

CaixaEletronico caixa = **new** CaixaEletronico();

**final** ServicoRemoto servicoRemotoMock = ctx.mock(ServicoRemoto.**class**);

caixa.setServicoRemoto(servicoRemotoMock);

String numeroContaTeste = "123456";

ContaCorrente contaCorrente = **new** ContaCorrente(numeroContaTeste);

ctx.checking(**new** Expectations() {{

oneOf(servicoRemotoMock).recuperarConta(numeroContaTeste);

will(*returnValue*(contaCorrente));

}});

*assertEquals*("Usuário Autenticado", caixa.logar(numeroContaTeste));

*assertEquals*("Retire seu dinheiro", caixa.sacar());

*assertEquals*("Depósito recebido com sucesso", caixa.depositar());

*assertEquals*("O saldo é R$xx,xx", caixa.saldo());

}

* 1. **teste – recuperar número de conta corrente inválida**

## Apenas para exercitar a mensagem de uma conta corrente não encontrada, foi criado o teste logarUsuarioNaoIdentificado, não foi necessário alterar nada no código do CaixaEletronico, apenas registrado o cenário de teste para futuros casos.

* Código de teste (source folder “test”)

@Test

**public** **void** logarUsuarioNaoIdentificado() {

CaixaEletronico caixa = **new** CaixaEletronico();

**final** ServicoRemoto servicoRemotoMock = ctx.mock(ServicoRemoto.**class**);

caixa.setServicoRemoto(servicoRemotoMock);

String numeroContaInvalido = "123";

ctx.checking(**new** Expectations() {{

oneOf(servicoRemotoMock).recuperarConta(numeroContaInvalido);

will(*returnValue*(**null**));

}});

*assertEquals*("Usuário não encontrado", caixa.logar(numeroContaInvalido));

}

## Analisando a similaridade entre os métodos de teste criados até o momento, percebeu-se que além do @Rule do JMock, o próprio CaixaEletronico estava sendo inicializado e configurado com um ServicoRemoto. Desta forma, foi realizada a refatoração para transformá-los em *fixtures* e inicializá-los num método @Before.

* Código de inicialização dos Testes (source folder “test”)

**private** **static** **final** String ***numeroContaTeste*** = "123456";

**private** CaixaEletronico caixa;

**private** ServicoRemoto servicoRemotoMock;

@Before

**public** **void** setUp() {

caixa = **new** CaixaEletronico();

servicoRemotoMock = ctx.mock(ServicoRemoto.**class**);

caixa.setServicoRemoto(servicoRemotoMock);

}

* 1. **teste – executar operações sem usuário logado**

## Aproveitando a estrutura do ServicoRemoto e recuperação da conta corrente, foi realizado um ciclo TDD para verificação das operações realizadas pelo CaixaEletronico sem usuário logado.

* Código de teste (source folder “test”)

@Test(expected = UsuarioNaoLogadoException.**class**)

**public** **void** executarOperacaoSaldoSemUsuarioLogado() {

String saldo = caixa.saldo();

*fail*("Operação não deveria ser executada sem nenhum usuário logado no caixa.");

}

@Test(expected = UsuarioNaoLogadoException.**class**)

**public** **void** executarOperacaoSacarSemUsuarioLogado() {

String sacar = caixa.sacar();

*fail*("Operação não deveria ser executada sem nenhum usuário logado no caixa.");

}

@Test(expected = UsuarioNaoLogadoException.**class**)

**public** **void** executarOperacaoDepositarSemUsuarioLogado() {

String depositar = caixa.depositar();

*fail*("Operação não deveria ser executada sem nenhum usuário logado no caixa.");

}

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

**package** br.inpe.cap.projetoagil;

**public** **class** CaixaEletronico {

**private** ServicoRemoto servicoRemoto;

**private** ContaCorrente contaCorrenteAtual;

**public** String logar(String numeroConta) {

**this**.contaCorrenteAtual = servicoRemoto.recuperarConta(numeroConta);

**if**(**this**.contaCorrenteAtual != **null**) {

**return** "Usuário Autenticado";

} **else** {

**return** "Usuário não encontrado";

}

**public** String sacar() {

**return** "Retire seu dinheiro";

}

**public** String depositar() {

**return** "Depósito recebido com sucesso";

}

**public** String saldo() {

**return** "O saldo é R$xx,xx";

}

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

**package** br.inpe.cap.projetoagil;

**public** **class** CaixaEletronico {

**private** ServicoRemoto servicoRemoto;

**private** ContaCorrente contaCorrenteAtual;

**public** String logar(String numeroConta) {

**this**.contaCorrenteAtual = servicoRemoto.recuperarConta(numeroConta);

**if**(**this**.contaCorrenteAtual != **null**) {

**return** "Usuário Autenticado";

} **else** {

**return** "Usuário não encontrado";

}

}

**public** String sacar() {

**if**(**this**.contaCorrenteAtual != **null**) {

**return** "Retire seu dinheiro";

} **else** {

**throw** **new** UsuarioNaoLogadoException();

}

}

**public** String depositar() {

**if**(**this**.contaCorrenteAtual != **null**) {

**return** "Depósito recebido com sucesso";

} **else** {

**throw** **new** UsuarioNaoLogadoException();

}

}

**public** String saldo() {

**if**(**this**.contaCorrenteAtual != **null**) {

**return** "O saldo é R$ xx,xx";

} **else** {

**throw** **new** UsuarioNaoLogadoException();

}

}

* 1. **teste – valor saldo correspondente**

## Foi criado o método de teste valorSaldoCorrespondente a fim de testar a lógica da recuperação e mensagem do valor do saldo:

* Código de teste (source folder “test”)

@Test

**public** **void** valorDoSaldoCorrespondente() {

ContaCorrente contaCorrente = **new** ContaCorrente(***numeroContaTeste***);

contaCorrente.setSaldo(252.50);

ctx.checking(**new** Expectations() {{

oneOf(servicoRemotoMock).recuperarConta(numeroContaTeste);

will(*returnValue*(contaCorrente));

}});

*assertEquals*("O saldo é R$252,50", caixa.saldo());

}

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

**public** String saldo() {

**if**(**this**.contaCorrenteAtual != **null**) {

**return** "O saldo é R$ xx,xx";

} **else** {

**throw** **new** UsuarioNaoLogadoException();

}

}

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

**private** **static** **final** NumberFormat ***SALDO\_FORMAT*** =

NumberFormat.*getCurrencyInstance*(**new** Locale("pt", "BR"));

**public** String saldo() {

**if**(**this**.contaCorrenteAtual != **null**) {

**double** saldo = **this**.contaCorrenteAtual.getSaldo();

**return** String.*format*("O saldo é %1s", ***SALDO\_FORMAT***.format(saldo));

} **else** {

**throw** **new** UsuarioNaoLogadoException();

}

}

## A execução do teste resultou em sucesso, porém o 1º teste passou a não funcionar mais por conta do saldo de uma ContaCorrente vazia ser 0,00. Portanto, o 1º teste foi ajustado também para o valor correspondente.

* Código de teste (source folder “test”) – ***Antes***

@Test

**public** **void** exibicaoMensagensCaixaEletronico() {

ContaCorrente contaCorrente = **new** ContaCorrente(***numeroContaTeste***);

ctx.checking(**new** Expectations() {{

oneOf(servicoRemotoMock).recuperarConta(***numeroContaTeste***);

will(*returnValue*(contaCorrente));

}});

*assertEquals*("Usuário Autenticado", caixa.logar(***numeroContaTeste***));

*assertEquals*("Retire seu dinheiro", caixa.sacar());

*assertEquals*("Depósito recebido com sucesso", caixa.depositar());

*assertEquals*("O saldo é R$xx,xx", caixa.saldo());

}

* Código de teste (source folder “test”) – ***Depois***

@Test

**public** **void** exibicaoMensagensCaixaEletronico() {

CaixaEletronico caixa = **new** CaixaEletronico();

**final** ServicoRemoto servicoRemotoMock = ctx.mock(ServicoRemoto.**class**);

caixa.setServicoRemoto(servicoRemotoMock);

ContaCorrente contaCorrente = **new** ContaCorrente(***numeroContaTeste***);

ctx.checking(**new** Expectations() {{

oneOf(servicoRemotoMock).recuperarConta(***numeroContaTeste***);

will(*returnValue*(contaCorrente));

}});

*assertEquals*("Usuário Autenticado", caixa.logar(***numeroContaTeste***));

*assertEquals*("Retire seu dinheiro", caixa.sacar());

*assertEquals*("Depósito recebido com sucesso", caixa.depositar());

*assertEquals*("O saldo é R$ 0,00", caixa.saldo());

}

* 1. **teste – operação de saque com sucesso**

## Foi criado o método de teste operacaoDeSaqueComSucesso a fim de implementar a funcionalidade de saque. Foi alterada a assinatura do método CaixaEletronico.sacar() com a inclusão de um parâmetro double correspondente ao valor de saque solicitado pelo usuário, além de criado método debitar() na classe ContaCorrente:

* Código de teste (source folder “test”)

@Test

**public** **void** operacaoDeSaqueComSucesso() {

ContaCorrente contaCorrente = **new** ContaCorrente(***numeroContaTeste***);

contaCorrente.setSaldo(252.50);

ctx.checking(**new** Expectations() {{

oneOf(servicoRemotoMock).recuperarConta(***numeroContaTeste***);

will(*returnValue*(contaCorrente));

}});

*assertEquals*("Usuário Autenticado", caixa.logar(***numeroContaTeste***));

*assertEquals*("Retire seu dinheiro", caixa.sacar(50));

*assertEquals*("O saldo é R$ 202,50", caixa.saldo());

}

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

**public** String sacar() {

**if**(**this**.contaCorrenteAtual != **null**) {

**return** "Retire seu dinheiro";

} **else** {

**throw** **new** UsuarioNaoLogadoException();

}

}

* Classes de domínio (source folder “src”) – ***Depois***
  + CaixaEletronico

**public** String sacar() {

**if**(**this**.contaCorrenteAtual != **null**) {

**if**(**this**.contaCorrenteAtual.getSaldo() >= valorSacado) {

**this**.contaCorrenteAtual.debitar(valorSacado);

**return** "Retire seu dinheiro";

} **else** {

**return** "Saldo insuficiente";

}

} **else** {

**throw** **new** UsuarioNaoLogadoException();

}

}

* + ContaCorrente

**public** **void** debitar(**double** valorSacado) {

**this**.saldo = saldo - valorSacado;

}

## Para confirmar a lógica implementada no método CaixaEletronico.sacar(), foi criado também o método de teste operacaoDeSaqueComSaldoInsuficiente:

* Código de teste (source folder “test”)

@Test

**public** **void** operacaoDeSaqueComSaldoInsuficiente() {

ContaCorrente contaCorrente = **new** ContaCorrente(***numeroContaTeste***);

contaCorrente.setSaldo(252.50);

ctx.checking(**new** Expectations() {{

oneOf(servicoRemotoMock).recuperarConta(***numeroContaTeste***);

will(*returnValue*(contaCorrente));

}});

*assertEquals*("Usuário Autenticado", caixa.logar(***numeroContaTeste***));

*assertEquals*("Saldo insuficiente", caixa.sacar(350));

*assertEquals*("O saldo é R$ 252,50", caixa.saldo());

}

* 1. **teste – operação de depósito com sucesso**

## Foi criado o método de teste operacaoDeDeposito a fim de implementar a funcionalidade de depósito. Foi alterada a assinatura do método CaixaEletronico.depositar() com a inclusão de um parâmetro double correspondente ao valor de depósito realizado pelo usuário, além de criado método creditar() na classe ContaCorrente:

* Código de teste (source folder “test”)

@Test

**public** **void** operacaoDeDeposito() {

ContaCorrente contaCorrente = **new** ContaCorrente(numeroContaTeste);

contaCorrente.setSaldo(252.50);

ctx.checking(**new** Expectations() {{

oneOf(servicoRemotoMock).recuperarConta(numeroContaTeste);

will(returnValue(contaCorrente));

}});

assertEquals("Usuário Autenticado", caixa.logar(numeroContaTeste));

assertEquals("Depósito recebido com sucesso", caixa.depositar(200));

assertEquals("O saldo é R$ 452,50", caixa.saldo());

}

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

**public** String depositar() {

**if**(**this**.contaCorrenteAtual != **null**) {

**return** "Depósito recebido com sucesso";

} **else** {

**throw** **new** UsuarioNaoLogadoException();

}

}

* Classes de domínio (source folder “src”) – ***Depois***
  + CaixaEletronico

**public** String depositar(**double** valorDepositado) {

**if**(**this**.contaCorrenteAtual != **null**) {

**this**.contaCorrenteAtual.creditar(valorDepositado);

**return** "Depósito recebido com sucesso";

} **else** {

**throw** **new** UsuarioNaoLogadoException();

}

}

* + ContaCorrente

**public** **void** creditar(**double** valorDepositado) {

**this**.saldo = saldo + valorDepositado;

}

* 1. **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 ;

}

}

* 1. **teste – Nome Composto (Separado por sublinhado)**

## Para testar o caso de um nome composto separado por sublinhado (foge da especificação CamelCase, porém utliza 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 ;

}

}

* 1. **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());

}

}

* 1. **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());

}

}

* 1. **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());

}

}

* 1. **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());

}

}

* 1. **teste – apenas conector e**

## No teste da palavra ser apenas a preposição “e”, ela não deve ser tratava 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());

}

}

* 1. **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.");

}

}

}

* 1. **teste – Palavra inválida contendo caractere especial**

## No teste de uma palavra inválida contendo caracteres especiais, inicialmente diversas expressões regulares foram testadas para validar caracteres não alfanuméricos, porém todas sem sucesso:

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

## Entretanto, o problema não estava na expressão regular utilizada, e sim no método utilizado para detecção de números e caracteres especiais. Havia sido utilizado o método *PATTERN*.matcher(original).matches(), que considera a String total passada como parâmetro. Como nenhuma das expressões testadas conseguem englobar toda a palavra, foi utilizado então o método *PATTERN*.matcher(original).find(), que percorre a String caractere por caracter, ou substring por substring em busca da expressão regular. Desta forma, a *regex* \W foi suficiente para fazer o teste passar, uma vez que ela retorna caracteres não utilizados em palavras.

* Código de teste (source folder “test”)

@Test(expected=IllegalArgumentException.**class**)

**public** **void** palavraInvalidaContendoCaracteresEspeciais() {

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).find()) {

**throw** **new** IllegalArgumentException(

"Caracteres especiais não são permitidos, somente letras e números.");

}

}