**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** "Não foi possível autenticar o usuário";

}

}

## 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*("Não foi possível autenticar o usuário", 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. **inclusão da interface de hardware no caixa eletrônico**

## A introdução da interface Hardware foi como uma evolução de requisitos, pois foi necessário remodelar o design da classe de domínio CaixaEletronico, alterando a assinatura do método logar retirando o parâmetro numeroConta, pois este agora seria recuperado pelo Hardware configurado no CaixaEletronico.

## Dessa forma, foi também incluído o mock correspondente ao Hardware, realizados todos os ajustes nas assinaturas e inclusão da cláusula throw HardwareMalfunctionException nos métodos de testes. Foram executados todos os testes para verificar se a refatoração não alterou o comportamento do CaixaEletronico antes de iniciar novo ciclo TDD com a introdução de novos testes simulando falha de hardware.

Como foi incluída a linha para pegar o número do cartão pelo Hardware, todos os testes falharam por falta de Expectation do mock do Hardware. Após as devidas adições de Expectation nos métodos de testes, todos os testes voltaram a funcionar normalmente.

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

**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);

}

@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(0));

*assertEquals*("Depósito recebido com sucesso", caixa.depositar(0));

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

}

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

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

**private** CaixaEletronico caixa;

**private** ServicoRemoto servicoRemotoMock;

**private** Hardware hardwareMock;

@Before

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

caixa = **new** CaixaEletronico();

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

caixa.setServicoRemoto(servicoRemotoMock);

hardwareMock = ctx.mock(Hardware.**class**);

caixa.setHardware(hardwareMock);

}

@Test

**public** **void** exibicaoMensagensCaixaEletronico() **throws** HardwareMalfunctionException {

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

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

oneOf(hardwareMock).pegarNumeroDaContaCartao();

will(*returnValue*(***numeroContaTeste***));

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

will(*returnValue*(contaCorrente));

}});

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

*assertEquals*("Retire seu dinheiro", caixa.sacar(0));

*assertEquals*("Depósito recebido com sucesso", caixa.depositar(0));

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

}

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

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

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

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

**return** "Usuário Autenticado";

} **else** {

**return** "Não foi possível autenticar o usuário";

}

}

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

**public** String logar() **throws** HardwareMalfunctionException {

String numeroConta = hardware.pegarNumeroDaContaCartao();

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

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

**return** "Usuário Autenticado";

} **else** {

**return** "Não foi possível autenticar o usuário";

}

}

* 1. **teste – operações com falha de hardware – login e depósito**

## Após o ciclo de TDD da introdução da interface Hardware, foram então incluídos os requisitos referentes ao Hardware e seus respectivos casos de testes e Expectations, implementando as chamadas dos métodos no hardware e simulando as falhas em cada operação:

* Código de teste (source folder “test”)

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

**public** **void** tentarEfetuarLoginComFalhaDeHardware() **throws** HardwareMalfunctionException {

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

oneOf(hardwareMock).pegarNumeroDaContaCartao();

will(*throwException*(**new** HardwareMalfunctionException()));

never(servicoRemotoMock).recuperarConta(***numeroContaTeste***);

}});

caixa.logar();

*fail*("Falha de hardware não deveria permitir realizar operações.");

}

@Test

**public** **void** operacaoDeDepositoComFalhaDeHardware() **throws** HardwareMalfunctionException {

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

contaCorrente.setSaldo(252.50);

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

oneOf(hardwareMock).pegarNumeroDaContaCartao();

will(*returnValue*(***numeroContaTeste***));

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

will(*returnValue*(contaCorrente));

oneOf(hardwareMock).lerEnvelope();

will(*throwException*(**new** HardwareMalfunctionException()));

}});

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

**try** {

caixa.depositar(200);

*fail*("Falha de hardware não deveria permitir realizar operações.");

} **catch** (HardwareMalfunctionException e) {

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

}

}

* 1. **teste – operações com falha de hardware - saque**

## No teste de saque com falha de hardware, foi bem interessante o fato de identificar um bug na ordem em que foi adicionado o comando para debitar o valor sacado do saldo do usuário. Após identificado esse bug, foi corrigido o código na classe de domínio, o ciclo TDD foi finalizado com sucesso do caso de teste.

* Código de teste (source folder “test”)

@Test

**public** **void** operacaoDeSaqueComFalhaDeHardware() **throws** HardwareMalfunctionException {

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

contaCorrente.setSaldo(252.50);

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

oneOf(hardwareMock).pegarNumeroDaContaCartao();

will(*returnValue*(***numeroContaTeste***));

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

will(*returnValue*(contaCorrente));

oneOf(hardwareMock).entregarDinheiro();

will(*throwException*(**new** HardwareMalfunctionException()));

}});

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

**try** {

caixa.sacar(50);

*fail*("Falha de hardware não deveria permitir realizar operações.");

} **catch** (HardwareMalfunctionException e) {

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

}

}

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

**public** String sacar(**double** valorSacado) **throws** HardwareMalfunctionException {

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

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

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

**this**.hardware.entregarDinheiro();

**return** "Retire seu dinheiro";

} **else** {

**return** "Saldo insuficiente";

}

} **else** {

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

}

}

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

**public** String sacar(**double** valorSacado) **throws** HardwareMalfunctionException {

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

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

**this**.hardware.entregarDinheiro();

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

**return** "Retire seu dinheiro";

} **else** {

**return** "Saldo insuficiente";

}

} **else** {

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

}

}