Desenvolvimento de Software de Caixa Eletrônico em Java com Uso de Objetos Mock

Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

I - INTRODUÇÃO

Este é o meu **Relatório** da **TAREFA da SEMANA TRÊS** do **Curso de Desenvolvimento Direcionado à Testes (TDD) da Plataforma Coursera em parceria com o ITA,** contendo ***descrição e detalhamento do projeto e de como foram atendidas todas às recomendações e requisitos*** especificados no documento "***Tarefa avaliada por colega: Software de Caixa Eletrônico***".

II – REQUISITOS PARA ESTA TAREFA FINAL DA SEMANA TRÊS:

As especificações providas foram de criar, utilizando TDD, uma classe chamada *CaixaEletronico*, juntamente com a classe *ContaCorrente*, que possuem os requisitos abaixo:

- A classe *CaixaEletronico* possui os métodos *logar(), sacar(), depositar() e saldo()* e todas retornam uma *String* com a mensagem que será exibida na tela do caixa eletrônico;
- Existe uma classe chamada *ContaCorrente* que possui as informações da conta necessárias para executar as funcionalidades do *CaixaEletronico*. Essa classe faz parte da implementação e deve ser definida durante a sessão de TDD;
- As informações da classe *ContaCorrente* podem ser obtidas utilizando os métodos de uma interface chamada *ServicoRemoto*. Essa interface possui o método *recuperarConta()* que recupera uma conta baseada no seu número e o método *persistirConta()* que grava alterações, como uma mudança no saldo devido a um saque ou depósito. Não tem nenhuma implementação disponível da interface *ServicoRemoto* e deve ser utilizado um *Mock Object* para ela durante os testes;
- O método *persistirConta()* da interface *ServicoRemoto* deve ser chamado **apenas** no caso de ser feito algum saque ou depósito **com sucesso;**
- Ao executar o método *saldo()*, a mensagem retornada deve ser "*O saldo é R$xx,xx*" com o valor do saldo;
- Ao executar o método *sacar(),* e a execução for com sucesso, deve retornar a mensagem "*Retire seu dinheiro*". Se o valor sacado for maior que o saldo da conta, a classe *CaixaEletronico* deve retornar uma *String* dizendo "*Saldo insuficiente";*
- Ao executar o método *depositar(),* e a execução for com sucesso, deve retornar a mensagem "Depósito recebido com sucesso";
- Ao executar o método *login()*, e a execução for com sucesso, deve retornar a mensagem "*Usuário Autenticado*". Caso falhe, deve retornar "*Não foi possível autenticar o usuário*";
- Existe uma interface chamada Hardware que possui os métodos *pegarNumeroDaContaCartao()* para ler o número da conta do cartão para o login (retorna uma *String* com o número da conta), *entregarDinheiro()* que entrega o dinheiro no caso do saque (*retorna void)* e *lerEnvelope()* que recebe o envelope com dinheiro na operação de depósito *(retorna void).* Não tem nenhuma implementação disponível da interface *Hardware* e deve ser utilizado um *Mock Object* para ela durante os testes.
- Todos os métodos da interface *Hardware* podem lançar uma *exceção* dizendo que *houve uma falha de funcionamento do hardware.*

Deve-se criar *testes* também para os *casos de falha*, principalmente na *classe Hardware* que pode falhar a qualquer momento devido a um mau funcionamento. Lembre-se de usar o TDD e ir incrementando as funcionalidades aos poucos. Você deve entregar o *código final*, incluindo os *testes* e os *mock objects* criados. Coloque todo *código relativo a teste em uma pasta separada.*

Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

III – DIAGRAMAS DE CASOS DE USO DO SOFTWARE DE CAIXA ELETRÔNICO:

Com base nas especificações passadas e de forma a atender todos os requisitos impostos para o desenvolvimento do *Software de Caixa Eletrônico*, inferi e produzi o *Diagrama UML de Casos de Usos* que contém os requisitos das funcionalidades requeridas, respectivamente:

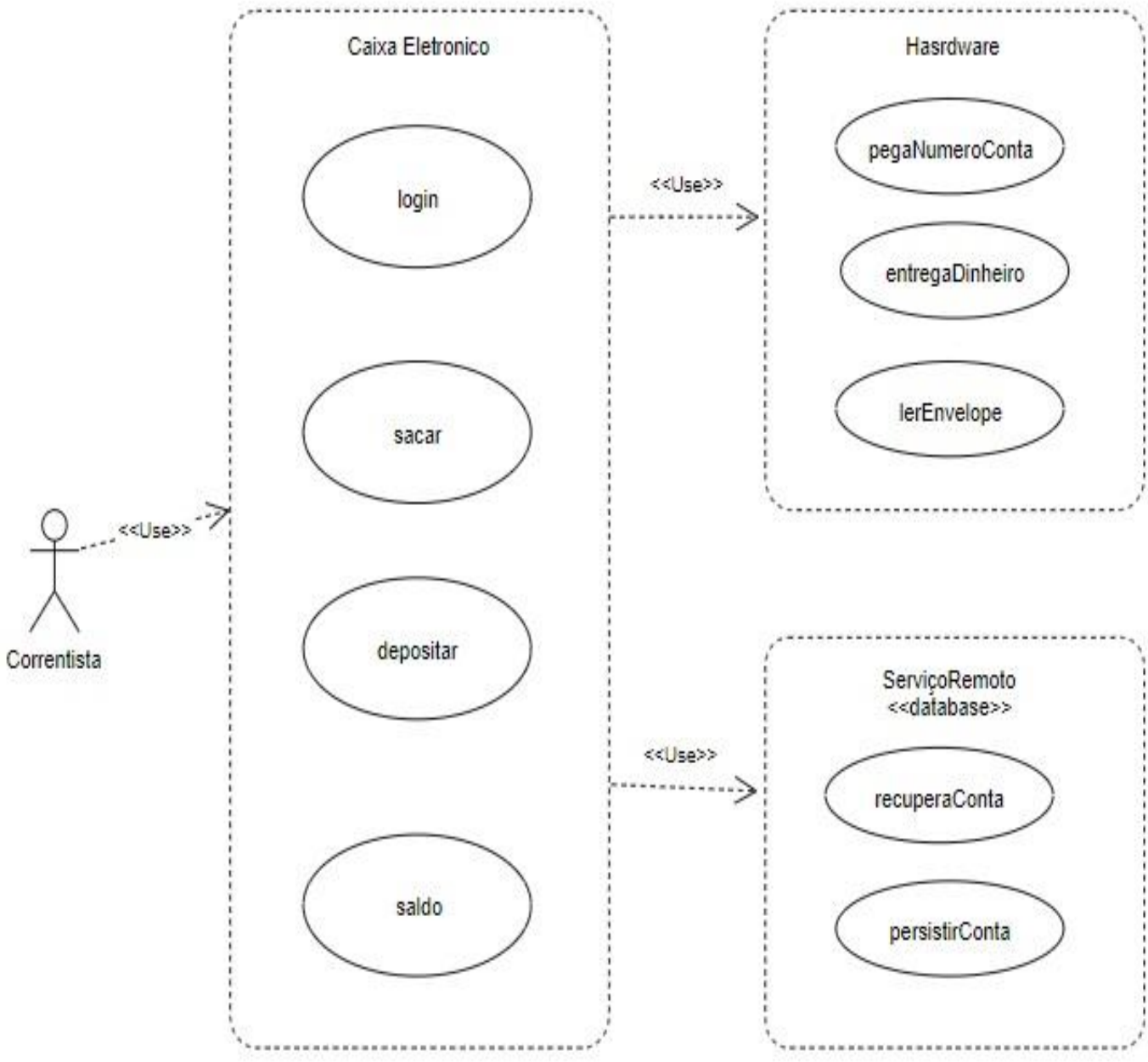


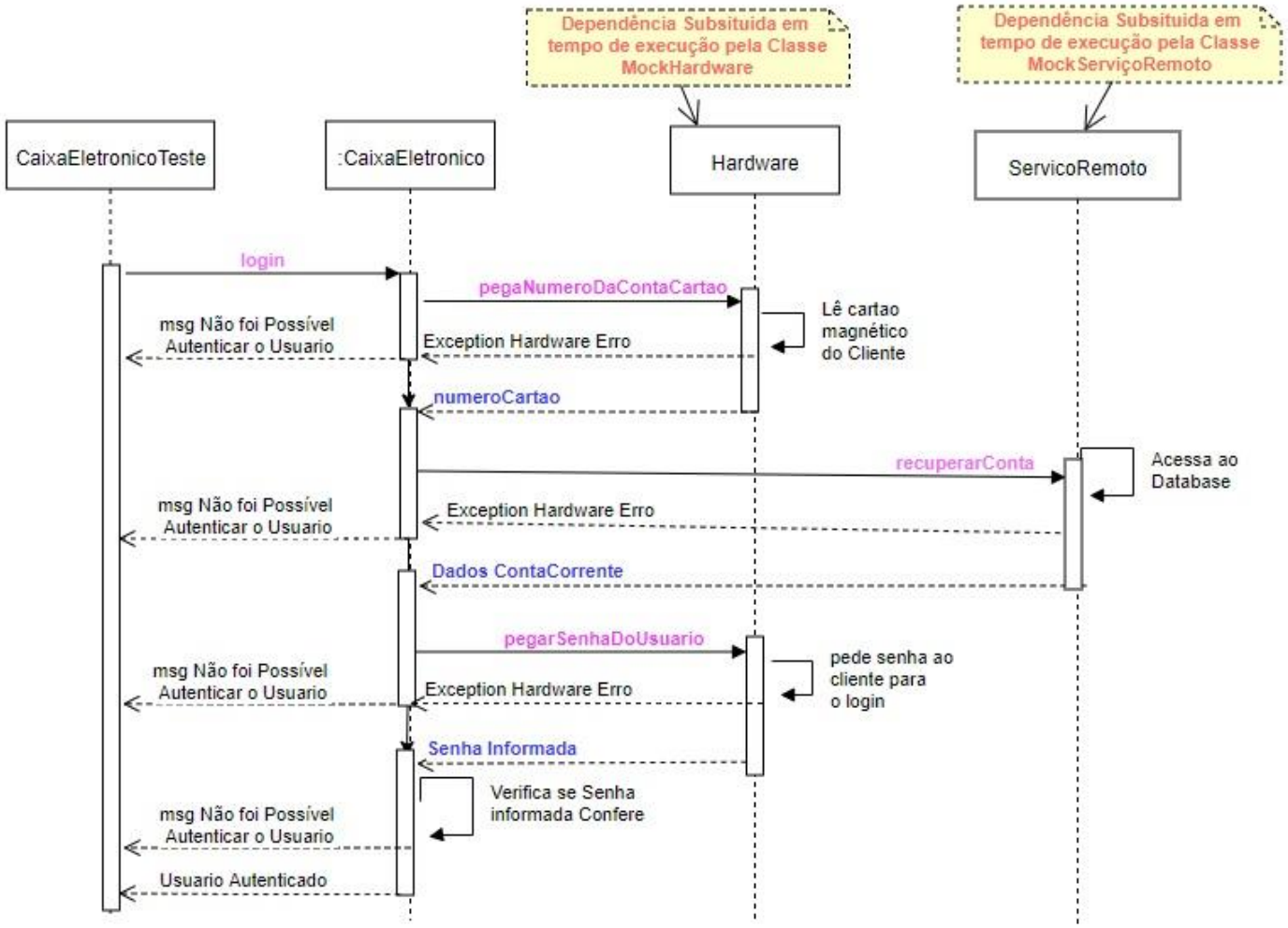
Figura 1 – Diagrama de Use Cases UML do Software de Caixa Eletrônico

Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

IV.1 – A Funcionalidade Login da CLASSE CaixaEletronico:

Com base na especificação provida pela Plataforma Cousera/ITA para a TAREFA da Semana três para desenvolvimento do Software de Caixa Eletrônico, foi possível inferir o *Diagrama UML de Sequência* para a Funcionalidade de LOGIN do Caixa Eletrônico, e incorporei a sequência de operações com base na Classe de Teste da Classe Caixa Eletrônico e mais especificamente no workflow referente ao processo do login respectivamente:

Figura 2 – Diagrama UML de Sequência do Use Case Login do Software de Caixa Eletrônico VERSÃO 3



O diagrama UML acima demonstra a sequência das operações do processo de login do correntista no Caixa Eletrônico estritamente de acordo com o especificado pela Platafomre Coursera/ITA, respectivamente:

- *O método login(0 da Classe CaixaEletronico ao ser invocado faz uma chamada ao método pegaNumeroDoCartao do Interface Hardware – que no nosso caso deverá ser atendido pelo Objeto Mock - e que este método pode devolver duas respostas de retorno ao método login que pode se: um exception de erro no hardware durante a leitura do cartão magnético do cliente, ou se a leitura for bem sucedida, devolver o número da conta lido do dispositivo magnético. Por sua vez o método login deve repassar para o método chamante um string com o número da conta ou a exception para sinalizar a operação mau sucedida do hardware na leitura do cartão do cliente;*

- *No caso o módulo chamante do método login da Classe CaixaEletrônico após ter retorno com sucesso do Login irá cahmar o método recuperaConta do ServiçoRemoto para obter os dados do ContaCorrente do cliente, podendo receber de resposta uma exception por erro no acesso do Banco de Dados ou devolver o objeto ContaCorrente com os dados de conta, senha e ;*

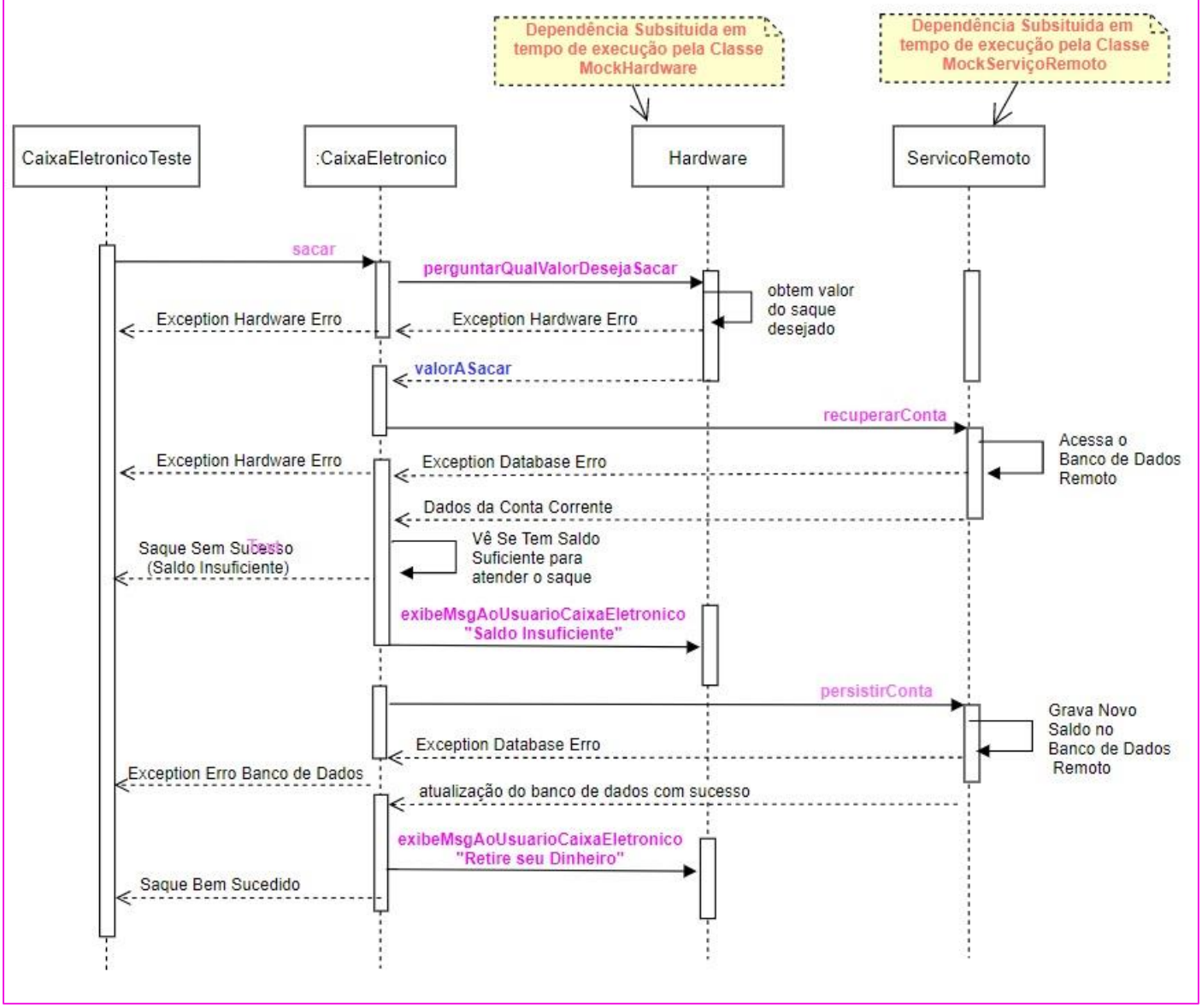
Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

- *Ao ser exewcutado o método login() e a execuação for com sucesso, este método deve retornar a msg "Usuário Autenticado". Caso Falhe dev retonar a msg "Não foi possível autenticar o usuário";*

IV.2 – A Funcionalidade Sacar da CLASSE CaixaEletronico:

Com base na especificação provida pela Plataforma Cousera/ITA para a TAREFA da Semana três para desenvolvimento do Software de Caixa Eletrônico, pude inferir o Diagrama UML de Sequência para o Caso do Usuário referente a operação SACAR no Caixa Eletrônico, respectivamente:

Figura 3 – Diagrama UML de Sequência do Use Case Login do Software de Caixa Eletrônico VERSÃO 3

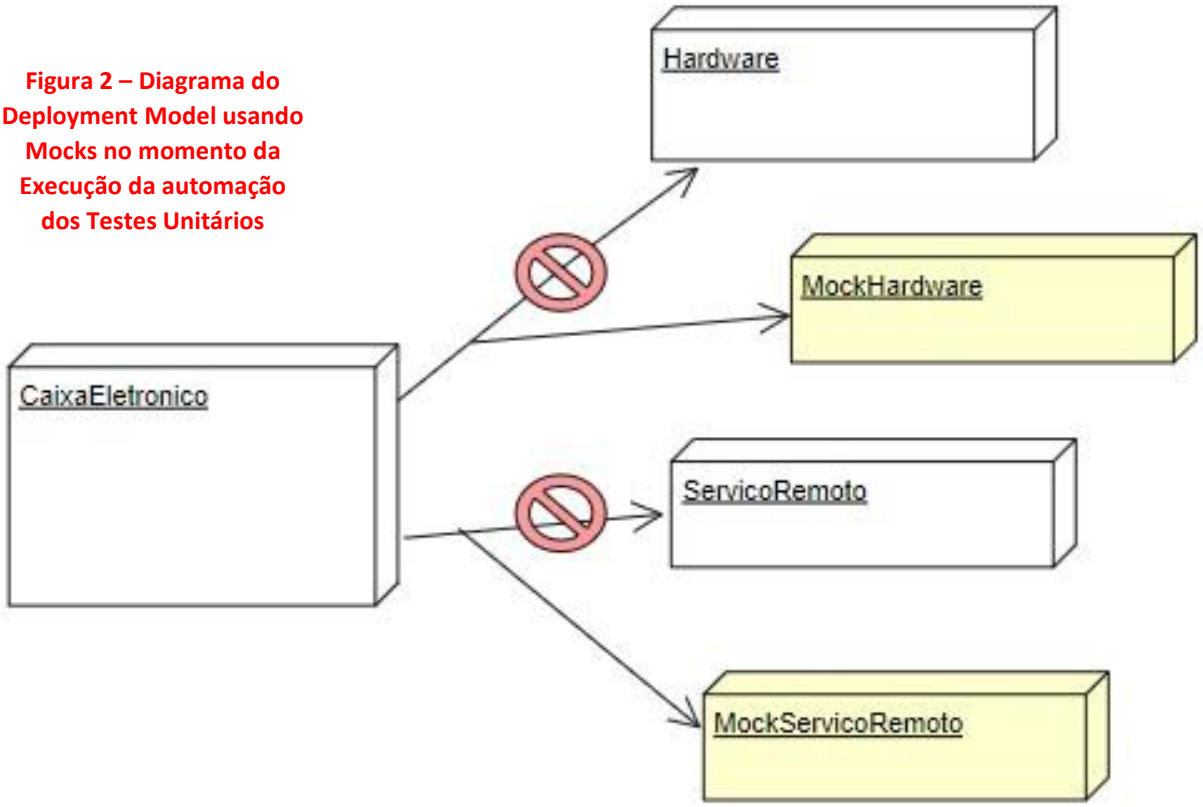


Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

IV – SUBSTITUIÇÃO DAS DEPENDÊNCIAS EXTERNAS POR MOCKS NOS TESTES UNITÁRIOS DO SOFTWARE DE CAIXA ELETRÔNICO:

As ***dependências externas*** da *Classe Caixa Eletrônico* foram substituídas por ***Objetos Mock*** em atendimento às exigências da ***tarefa da Semana Três da Plataforma Coursera/ITA*** durante o uso da ***técnica TDD (Test Driven Development)*** para desenvolvimento do Software de Caixa Eletrônico. Esta substituição deve-se, também, às recomendações técnicas, devido ao fato de que, no ***processo de automação dos Casos de Testes Unitários da Classe Caixa Eletrônico*** *d*evo isolar esta classe das outras classes das quais ela depende. Isso decorre do fato de que, não estarei fazendo **testes integrados**, mas sim, desejo realizar **somente testes unitários** de modo a assegurar que as funcionalidades específicas da Classe Caixa Eletrônico estejam funcionando conforme especificado, e portanto, atendam aos ***testes de Caixa Preta***, isto é, os ***testes funcionais da aplicação***. Assim, terei a seguinte situação:

Figura 2 – Diagrama do Deployment Model usando Mocks no momento da Execução da automação dos Testes Unitários



Para **implementação dos MOCKS** em substituição às **dependências externas** representadas pelos **Interfaces:**

- **Hardware** (*abstração responsável pela interação com o hardware específico do caixa eletrônico*), e
- **Serviço Remoto** *(abstração responsável em atender o acesso ao banco de dados externo onde estão armazenadas as contacorrentes bancárias*);

Utilizei o ***Padrão de Projeto*** (**Design Pattern**) denominado ***INJEÇÃO DE DEPENDÊNCIA*** (**Dependency Injection**) para substituir os mocks no lugar das dependências externas.

Esta **técnica** é a maneira utilizada com o fim de permitir que o acoplamento entre a classe sendo testada e suas dependências seja quebrada durante a automação dos testes unitários.

Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

Há ***três maneiras*** de se implementar a ***Injeção de Dependência***:

- *por meio de injeção de parâmetros*, ou
- *por meio de injeção no construtor*, ou
- *por meio de injeção de atributo em um método set*.

A ***injeção de dependência*** é, assim, a forma de prover para a classe sendo testada qual o objeto a usar no lugar de cada dependência externa em tempo de run-time e durante a execução dos testes unitários.

Como optei por usar a ***Linguagem JAVA*** para desenvolver o *Software de Caixa Eletrônico*, e esta linguagem é do tipo *static binding*, pois ela requer que os tipos exatos ou classes sejam definidos antes do uso (antes de compilar) – isso limita severamente as opções relacionadas em como podemos configurar o software em tempo de execução – o que se contrapões às linguagens com *dynamic binding*, que são mais flexíveis e permitem postergar a decisão exata de qual tipo ou qual classe usar para o momento da execução (*run-time*).

A ***Injeção de Dependência*** é uma boa opção para informar qual classe usar quanto estamos projetando o software do zero – **que é o caso deste Software de Caixa Eletrônico**.

Segundo a literatura, a ***Injeção de Dependência*** oferece um meio natural de projetar o código da minha aplicação, principalmente, quando estou utilizando o *Teste-Driven Development (TDD)*, porque muitos dos nossos *casos de testes unitários*, que escrevi, para os *objetos dependentes*, busca substituir a classe da qual se depende com um *Test Doube*(no meu caso um *objeto Mock*).

Ainda, segundo a literatura técnica, qualquer que seja o mecanismo escolhido para fazer ***a injeção de dependência*** *(substituição da dependência externa pelo mock objeto)* na classe sendo testada (*Classe CaixaEletronico*), devo assegurar que o *Mock Objeto*, que usarei para substituir no lugar da dependência externa, ***seja tipo compatível com o código que usará o Test Double***.

Isso é facilmente feito se tanto o componente real a substituir quanto o objeto Mock implementam o mesmo interface. *Esta restrição está sendo atendida*, visto que a *Plataforma Coursera/ITA* na especificação do *Software de CaixaEletronico* a desenvolver já especificou duas interfaces para as duas classes que são dependências externas da Classe CaixaEletronico, respectivamente:

- ***interface Hardware*** e
- ***interface ServiçoRemoto***.

Logo, a utilização de **Mocks Objetos** é factível pelo **método de Injeção de Dependência** no desenvolvimento da aplicação de Caixa Eletrônico.

Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

V – ARQUITETURA DO PROJETO DO SOFTWARE DE CAIXA ELETRÔNICO:

Com base nas especificações passadas e de forma a atender todos os requisitos impostos para o desenvolvimento do ***Software de Caixa Eletrônico***, inferi e produzi o *Diagrama UML de Classes e seus Relacionamentos* que contém as classes e interfaces requeridos, respectivamente:

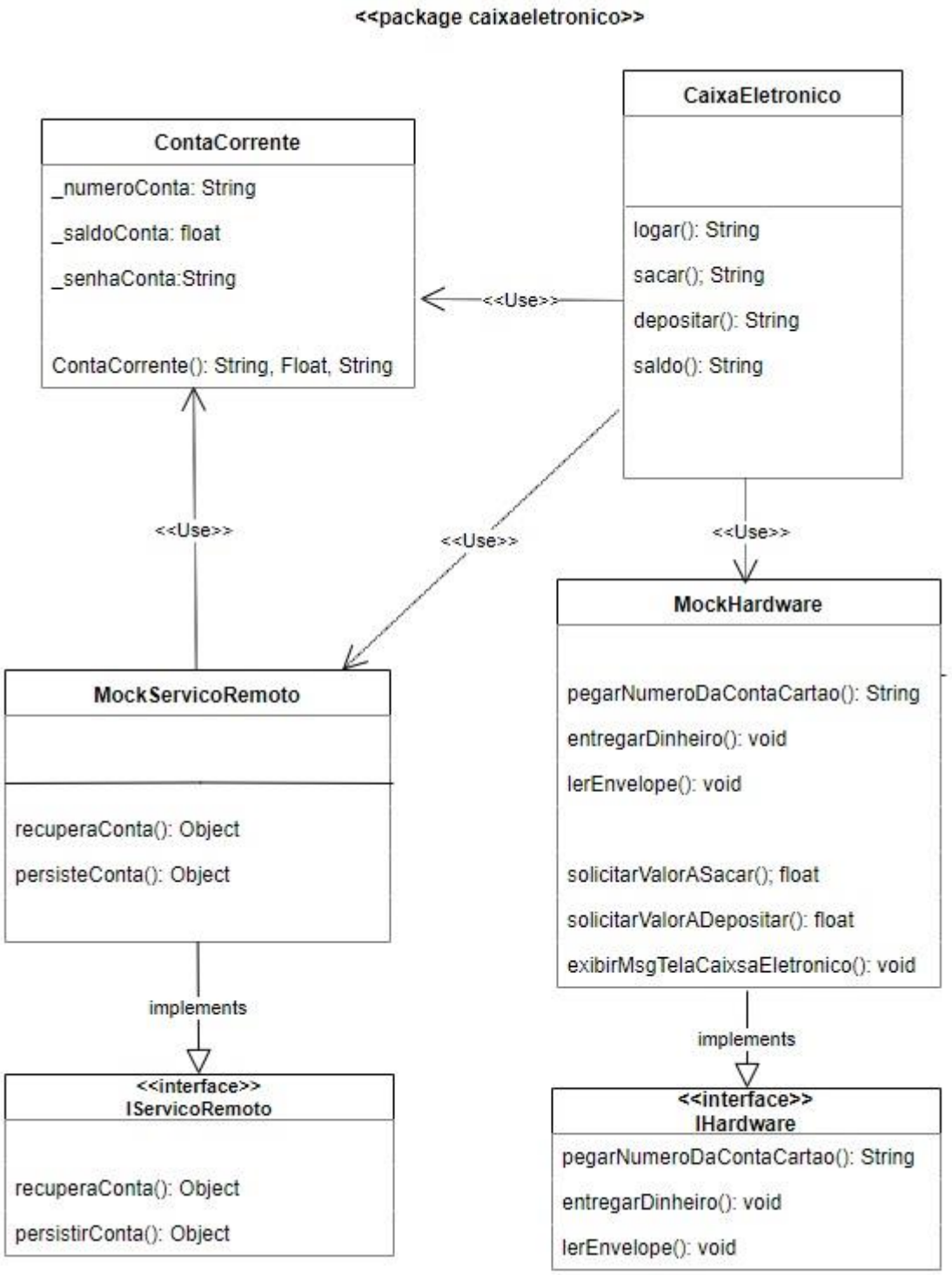


Figura 3 – Diagrama de Classes UML do Software de Caixa Eletrônico

Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

VI – O AMBIENTE UTILIZADO P/O DESENVOLVIMENTO DO SOFTWARE

Considerando que, não houve nenhuma exigência da *Plataforma Coursera/ITA* quanto a ***linguagem de programação*** a usar, nem do framework adotado para o desenvolvimento do ***Suite de Casos de Testes Unitários***, e nem da ***plataforma IDE*** utilizada, optei pelos seguintes recursos:

1 - *Linguagem de Programa Orientada à Objetos* ***Java*** *na* ***Versão 1.08***
2 - ***ECLIPSE IDE (****Integrated Development Environment)* ***Versão 2020-09***
3 - *Framework* ***JUnit Versão 4*** *para o desenvolvimento dos* ***Cases de Testes unitários****;*
4 – ***Framework PIT de Testes de Mutação para Java*** *via plug-in do* ***Eclipse 2020-09 -***

A Estrutura de Folders para o presente projeto Java se apresenta da seguinte maneira – observe que, em atendimento ao solicitado como requisito para este projeto:

- ***as classes de testes estão todas colocadas no*** folder test e
- ***as classes e interfaces normais estão no*** folder src:

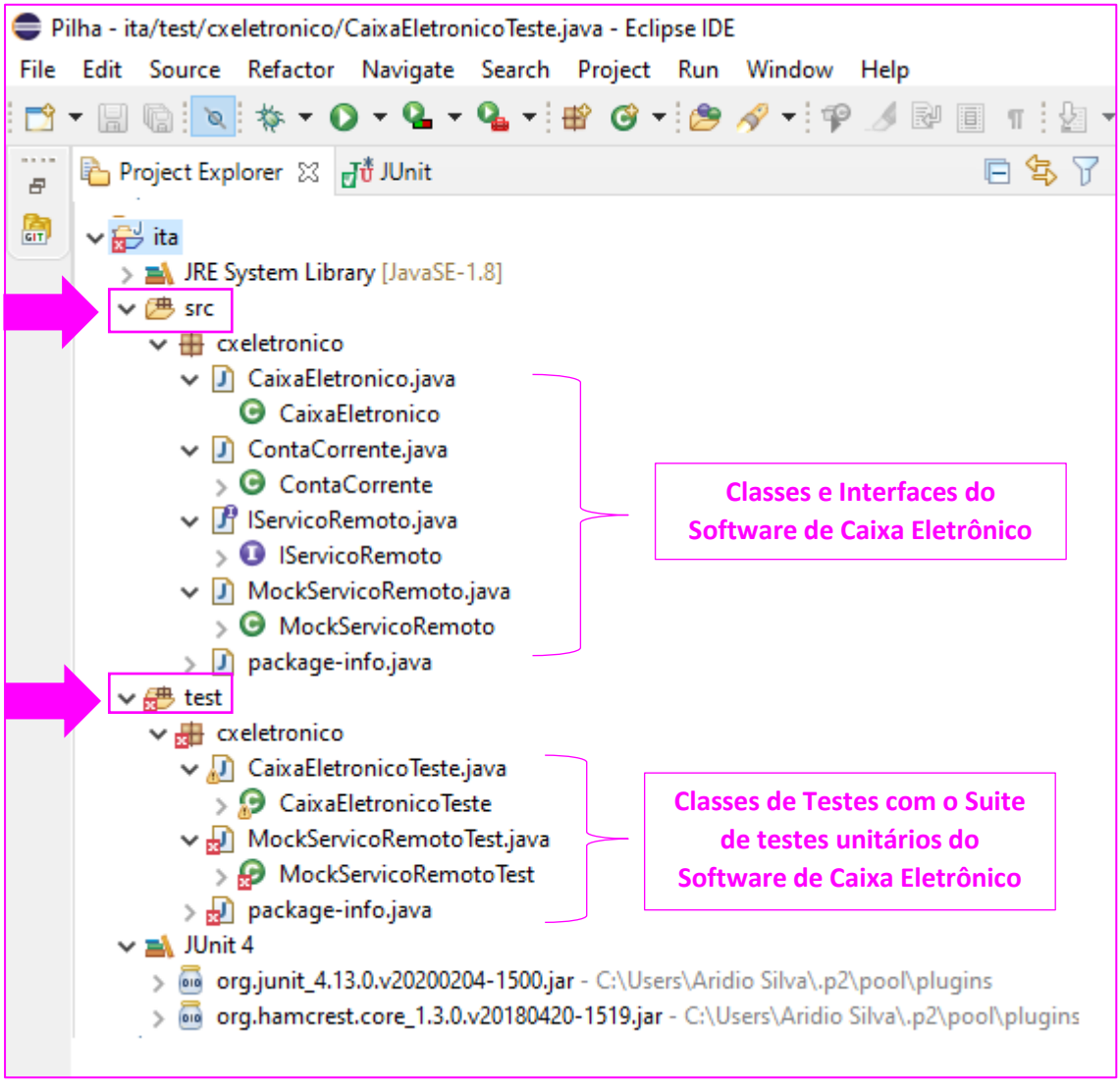


Figura 4 – Estrutura de Folders do Projeto JAVA

Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

VII – O CICLO DO TDD USADO NO DESENVOLVIMENTO DO SOFTWARE E OS TESTES UNITÁRIOS

Conforme pedido, durante o desenvolvimento do software guiado por testes, adotei e respeitei o *Ciclo do TDD*, sempre:

- *iniciando com o desenvolvimento de um teste que falha (e que cada caso de teste se baseia na especificação da API do método que implementa o requisito da aplicação correspondente);*
- *seguido da etapa de implementar o código de produção da responsabilidade associada (esta etapa persiste até que o teste unitário correspondente passe sem erro);*
- *seguido da etapa de refatoração do código de produção da produção desenvolvido até o momento (sem alterar as funcionalidades presentes e sem alterar os testes associados desenvolvidos).*

Este ciclo se repetiu para cada novo requisito implementado passo a passo da aplicação Caixa Eletrônico. Assim, o design da aplicação foi progressivamente e evolutivamente construído. A cada ciclo todos os casos de testes até o momento foram novamente rodados, de forma a garantir que nenhum acréscimo no código da aplicação tenha quadrado nenhuma outra parte do software do Caixa Eletrônico. Assim, tive a garantia de que, a aplicação Caixa Eletrônico foi desenvolvida com qualidade do início ao fim.

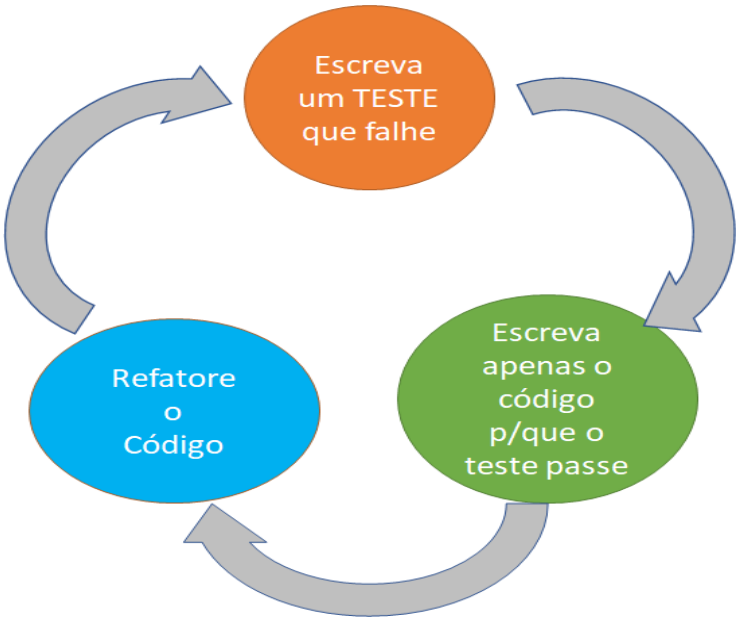


Figura 5

O CICLO do TDD

VIII – REALIZAÇÃO DE TESTES DE COBERTURA DO CÓDIGO DO SOFTWARE DESENVOLVIDO

Utilizei o *Teste de Cobertura do Código*, recurso disponível no *Eclipse IDE Versão 2020-09*, para verificar se todas as linhas do código escrito da aplicação Caixa Eletrônico, tanto do código de produção, quanto dos testes foram executados em sua plenitude e não ficou nenhuma parte sem ter sido exercitada.

IX – REALIZAÇAO DE TESTES DE MUTAÇÃO PARA AVALIAR A QUALIDADE DOS CASOS DE TESTES CRIADOS

Como complemento, utilizei o framework PIT para poder rodar testes de mutação para verificar a qualidade dos testes desenvolvidos.

X – PLANEJAMENTO DOS CASOS DE TESTES DO SOFTWARE DE CAIXA ELETRÔNICO:

Considerando que, a

Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

XI – CÓDIGO BASE FONTE DO SOFTWARE CAIXA ELETRONICO

A – PADRÃO DE ESTILO USADO NA PROGRAMAÇÃO DO SOFTWARE CAIXA ELETRONICO

- ***Padrao CamelCase*** – É a prática de escrever frases e palavras sem espaços entre elas e sem pontuação, onde a separação entre palavras se faz com uma primeira letra da palavra em maiúsculo e as demais em minúsculo, porém, dependendo do tipo do uso do nome tem algumas variações. No caso da programação Java o estilo usado consiste de palavras ou frases compostas tal modo que cada palavra da frase começa sempre com letra maiúscula, ou primeira letra minúscula, ou um símbolo, dependendo do objeto sendo nomeado, respectivamente:

    - ***Nomes de Classes*** – são substantivos, com letra maiúscula e minúscula mixada onde a primeira letra de cada palavra interna na frase é em maiúscula e as demais minúsculas;
    - ***Nomes de Interfaces*** – Segue o padrão dos nomes de classes, porem inicia sempre com uma Letra "I" maiúscula para diferenciar do nome das classes;
    - ***Nomes de Métodos*** - começa sempre com a primeira letra em minúsculo e a primeira letra das palavras subsequentes em maiúsculos e as demais letras em minúsculo;
    - ***Nomes de Variáveis de Instância*** – Os nomes de variáveis devem ter os nomes menores possíveis, porém devem passar a ideia da intenção do seu uso sem dúvidas – isto é, tenham significância mesmo sendo mnemônicas – o nome caso das variáveis de Instância de classes e Métodos, devem sempre iniciar sempre com "_" para diferenciá-las dos demais nomes;
    - ***Nomes de Argumentos de Métodos*** –
    - ***Nomes de Constantes*** – Devem ser sempre em Letra Maiúscula com as todas as palavras separadas por "_" (sublinhado);
    - ***Nomes de Packages*** – O prefixo de um package de nome de um top-level-domain sempre deve ter todas as letras em minúsculo, tipo .gov, .edu, .mil, .com, etc.; os componentes subsequentes do nome do pacote varia de acordo com o da organização própria de cada um (ou de cada empresa) – no meu caso adotei tudo em minúsculo para nome de pacotes;

- **DUMP – Descriptive And Meaningful Phrases** – **Frases Descritivas e com Significado** - Os nomes de classes, métodos, interfaces e classes de testes devem promover legibilidade e a compreensibilidade do código fonte – a finalidade de cada variável, de cada argumento, de cada nome de classe, de método, de interface, além do que cada parte do código faz. Lembrando que, para manter o código fonte, precisamos primeiro entendê-lo. Para entender o código, precisa ler e compreender. Considere por um momento quanto tempo você gastas lendo programas fontes e programas de testes? É muito tempo mesmo!!! Assim, a regra DAMP aumenta a manutenibilidade e legibilidade

Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

B– CÓDIGO FONTE DAS CLASSES DE TESTES UNITÁRIOS DO SOFTWARE DE CAIXA ELETRÔNICO

B.1 – Classe MockServicoRemotoTest

```
package cxeletronico;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class MockServicoRemotoTest {

	MockServicoRemoto _mock = new MockServicoRemoto();

	@Test
	public void recuperarPrimeiraContaCorrente( ) {
		_mock.persistirConta(new ContaCorrente ("9998", 100.0f, "XYZabc"));
		ContaCorrente _cc = _mock.recuperarConta("9998");
		assertEquals ("9998", _cc.getNumeroConta());
		assertEquals (100.0f, _cc.getSaldo());
		assertEquals ("XYZabc", _cc.getSenha());
	}
	@Test
	public void recuperarSegundaContaCorrente( ) {
		_mock.persistirConta(new ContaCorrente ("124578", 200.0f, "@CC"));
		ContaCorrente _cc = _mock.recuperarConta("124578");
		assertEquals ("124578", _cc.getNumeroConta());
		assertEquals (200.0f, _cc.getSaldo());
		assertEquals ("@CC", _cc.getSenha());
	}
	@Test
	public void recuperarTerceiraContaCorrente( ) {
		_mock.persistirConta(new ContaCorrente ("121212",  50.0f, "senha1"));
		ContaCorrente _cc = _mock.recuperarConta("121212");
		assertEquals ("121212", _cc.getNumeroConta());
		assertEquals (50.0f, _cc.getSaldo());
		assertEquals ("senha1", _cc.getSenha());
	}
	@Test
	public void recuperarQuartaContaCorrente( ) {
		_mock.persistirConta(new ContaCorrente ("232323",   0.0f, "senha33"));
		ContaCorrente _cc = _mock.recuperarConta("232323");
		assertEquals ("232323", _cc.getNumeroConta());
		assertEquals (0.0f, _cc.getSaldo());
		assertEquals ("senha33", _cc.getSenha());
	}
	@Test
	public void recuperarQuintaContaCorrente( ) {
		_mock.persistirConta(new ContaCorrente ("414141", 300.0f, "senha21"));
		ContaCorrente _cc = _mock.recuperarConta("414141");
		assertEquals ("414141", _cc.getNumeroConta());
		assertEquals (300.0f, _cc.getSaldo());
		assertEquals ("senha21", _cc.getSenha());
	}
	@Test
	public void recuperarSextaContaCorrente( ) {
		_mock.persistirConta(new ContaCorrente ("515151", 150.0f, "senha99"));
```

Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

```
		ContaCorrente _cc = _mock.recuperarConta("515151");
		assertEquals ("515151", _cc.getNumeroConta());
		assertEquals (150.0f, _cc.getSaldo());
		assertEquals ("senha99", _cc.getSenha());
	}
	@Test
	public void recuperarSetimaContaCorrente( ) {
		_mock.persistirConta(new ContaCorrente ("616161",  90.0f, "senha13"));
		ContaCorrente _cc = _mock.recuperarConta("616161");
		assertEquals ("616161", _cc.getNumeroConta());
		assertEquals (90.0f, _cc.getSaldo());
		assertEquals ("senha13", _cc.getSenha());
	}
	@Test
	public void recuperarOitavaContaCorrente( ) {
		_mock.persistirConta(new ContaCorrente ("646464", -450.0f, "senha66"));
		ContaCorrente _cc = _mock.recuperarConta("646464");
		assertEquals ("646464", _cc.getNumeroConta());
		assertEquals (-450.0f, _cc.getSaldo());
		assertEquals ("senha66", _cc.getSenha());
	}

}
```

Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

B.2 – Classe CaixaEletronicoTest

```
package cxeletronico;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class CaixaEletronicoTest {

	private MockHardware	_mockHD = new MockHardware ();
	private MockServicoRemoto _mockSR = new MockServicoRemoto();
	private CaixaEletronico	_cxe	= new CaixaEletronico ();

	@Before
	public void init() {
		_cxe.adicionaHardware(_mockHD);
		_cxe.adicionaServicoRemoto(_mockSR);
	}

	@Test
	public void testaPrimeiroLogin() {
		_mockSR.persistirConta(new ContaCorrente ("22222", 200.0f, "xyz"));
		ContaCorrente _cc = _mockSR.recuperarConta("22222");
		assertEquals ("22222", _cc.getNumeroConta());
		assertEquals (200.0f, _cc.getSaldo());
		assertEquals ("xyz", _cc.getSenha());
		assertEquals ( true, _cxe.login());
	}
	@Test
	public void testaSegundoLogin() {
		_mockSR.persistirConta(new ContaCorrente ("1234", 100.0f, "senha1"));
		ContaCorrente _cc = _mockSR.recuperarConta("1234");
		assertEquals ("1234", _cc.getNumeroConta());
		assertEquals (100.0f, _cc.getSaldo());
		assertEquals ("senha1", _cc.getSenha());
		assertEquals ( true, _cxe.login());
	}

}
```

Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

C – CÓDIGO FONTE DAS CLASSES E INTERFACES NORMAIS DO SOFTWARE DE CAIXA ELETRÔNICO

C.1 – Interface Hardware

```
package cxeletronico;

public interface IHardware {

	public String pegaNumeroDaConta ();
	public void entregarDinheiro (String msg);
	public void lerEnvelope (String msg);

	public String solicitaSenhaDoUsuario();
	public void exibeMsgAoUsuarioCaixaEletronico(String msg);

}
```

C.2 – Interface ServicoRemoto

```
package cxeletronico;

public interface IServicoRemoto {

	public ContaCorrente recuperarConta(String numeroConta);
	public void persistirConta(ContaCorrente cc);

}
```

C.2 – Classe ContaCorrente

```
package cxeletronico;

public class ContaCorrente {

	private String _numeroConta;
	private float _saldoConta;
	private String _senhaConta;

	public ContaCorrente (String numeroConta, float saldoConta, String senhaConta) {
		this._numeroConta = numeroConta;
		this._saldoConta = saldoConta;
		this._senhaConta = senhaConta;
	}

	public String getNumeroConta() {
		return _numeroConta;
	}

	public Object getSaldo() {
		return (float) _saldoConta;
	}

	public String getSenha() {
		return _senhaConta;
```

Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

```
	}
}
```

C.2 – Classe CaixaEletronico

```
package cxeletronico;

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

public class CaixaEletronico {

	private List<IHardware> _hardware = new ArrayList(1);
	private List<IServicoRemoto> _servRemoto  = new ArrayList(1);
	private IHardware _h;
	private IServicoRemoto _s;
	private ContaCorrente _cc;

	public Boolean login() {

		String _LOGIN_OK = "Usuario Autenticado com Sucesso";
		String _LOGIN_FALHOU = "Não foi possivel autenticar usuario";
		String _numeroConta;
		String _senhaUsuario;

		_h = _hardware.get(0);
		_s = _servRemoto.get(0);
		try {
			_numeroConta  = _h.pegaNumeroDaConta();
			_cc = _s.recuperarConta(_numeroConta);
			_senhaUsuario = _h.solicitaSenhaDoUsuario();
			String _senhaGravada = _cc.getSenha();
			if ( !_senhaUsuario.contains(_senhaGravada) ) {
			    _h.exibeMsgAoUsuarioCaixaEletronico(_LOGIN_FALHOU);
			    return false;
			}
		} catch (Exception e) {
			_h.exibeMsgAoUsuarioCaixaEletronico(_LOGIN_FALHOU);
			return false;
		}
		_h.exibeMsgAoUsuarioCaixaEletronico(_LOGIN_OK);
		return true;
	}

	public void adicionaHardware(IHardware hardwareCXE) {
		_hardware.add(hardwareCXE);
	}

	public void adicionaServicoRemoto(IServicoRemoto servRemoto) {
		_servRemoto.add(servRemoto);
	}

}
```

Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

C.5 – Classe MockServicoRemoto

```
package cxeletronico;

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

public class MockServicoRemoto implements IServicoRemoto {

	private static List<ContaCorrente> _contas = new ArrayList<>();

	@Override
	public ContaCorrente recuperarConta(String numeroConta) {

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

		    String _numConta = _contas.get(i).getNumeroConta();

		    if ( _numConta.contains(numeroConta) ) {

				ContaCorrente _ccItem = new ContaCorrente(
				                (String)_contas.get(i).getNumeroConta(),
				            (float) _contas.get(i).getSaldo(),
				                (String)_contas.get(i).getSenha()  );
			return _ccItem;
		    }
		}
		System.out.println ("Problema - Conta Corrente Não Existe >> " + numeroConta);
		throw new RuntimeException ("Problema - Conta Corrente Não Existe");

	}

	@Override
	public void persistirConta(ContaCorrente cc) {
		_contas.add(cc);
	}

	 public String devolveNumConta(int numRegistro) {

		if ( _contas.isEmpty()  || _contas.size() < numRegistro - 1)

			throw new RuntimeException ("Erro - Database Vazio!!!");

		return (String)_contas.get(numRegistro).getNumeroConta();
	 }

	 public String devolveSenhaConta(int numRegistro) {

		if ( _contas.isEmpty()  || _contas.size() < numRegistro - 1)

			throw new RuntimeException ("Erro - Database Vazio!!!");

		return (String)_contas.get(numRegistro).getSenha();
	 }

}
```

Aridio Gomes da Silva - aridiosilva@aridiosilva.com - https://www.linkedin.com/in/aridio-silva-74997111/

C.6 – Classe MockHardware

```
package cxeletronico;

import java.util.List;
import java.util.Random;

public class MockHardware implements IHardware {

	private static int _indice=-1;
	private int _INDEX_MAXIMO_CONTAS = 9;

	@Override
	public String pegaNumeroDaConta() {

		if (_indice > _INDEX_MAXIMO_CONTAS || _indice < 0)

			_indice = 0;	else	_indice++;

		System.out.println(" HARDWARE indice: " + _indice );
		MockServicoRemoto _mockSR = new MockServicoRemoto();
		return (String) _mockSR.devolveNumConta(_indice);
	}

	@Override
	public void entregarDinheiro(String msg) {

	}

	@Override
	public void lerEnvelope(String msg) {

	}

	public void exibeMsgAoUsuarioCaixaEletronico(String msg) {
		System.out.println("MSG: " + msg);
	}

	public String solicitaSenhaDoUsuario () {

		System.out.println(" HARDWARE indice: " + _indice );
		MockServicoRemoto _mockSR = new MockServicoRemoto();
		return (String) _mockSR.devolveSenhaConta(_indice);
	}

}
```