**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****.*

**III – MODELAGEM UML do SOFTWARE de CAIXA ELETRÔNICO:**

**III.1 – DIAGRAMAS de CASOS DE USO do SOFTWARE de CAIXA ELETRÔNICO:**

Com base nas especificações de requisitos passadas e de forma a atender todas as funcionalidades determinadas 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:

**Diagrama

Descrição gerada automaticamente**

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

**III.2 – DIAGRAMAS de SEQUÊNCIA e COLABORAÇÃO das CLASSES do SOFTWARE de CAIXA ELETRÔNICO:**

**III.2.1 – A Funcionalidade Login da CLASSE CaixaEletronico:**

A partir das **especificações da TAREFA da Semana três** providas pela **Plataforma Cousera/ITA** para o desenvolvimento Software de Caixa Eletrônico, foi possível elaborar o *Diagrama UML de Sequência,* que mostra a colaboração das classes na *Funcionalidade LOGIN do Caixa Eletrônico*, e respectivo workflow , incluindo a *Classe de Teste da Classe Caixa* Eletrônico e todo o workflow referente ao processo do login respectivamente:

Diagrama

Descrição gerada automaticamente

**Figura 2 – Diagrama UML de Sequência do Use Case Login do Caixa Eletrônico**

**III.2.2 – A Funcionalidade Sacar da CLASSE CaixaEletronico:**

A partir das **especificações de requisitos da TAREFA da Semana três** providas pela **Plataforma Cousera/ITA** para o desenvolvimento Software de Caixa Eletrônico, foi possível elaborar o *Diagrama UML de Sequência,* que mostra a colaboração das classes na *Funcionalidade SACAR do Caixa Eletrônico*, e toda a sequência de operações , incluindo a Classe de Teste da Classe Caixa Eletrônico e respectivo workflow referente ao processo do login respectivamente:

Diagrama

Descrição gerada automaticamente

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 Sacar do Caixa Eletrônico**

**III.2.3 – A Funcionalidade Depositar da CLASSE CaixaEletronico:**

A partir das **especificações de requisitos da TAREFA da Semana três** providas pela **Plataforma Cousera/ITA** para o desenvolvimento Software de Caixa Eletrônico, foi possível elaborar o *Diagrama UML de Sequência,* que mostra a colaboração das classes na *Funcionalidade DEPOSITAR do Caixa Eletrônico*, e toda a sequência de operações , incluindo a Classe de Teste da Classe Caixa Eletrônico e respectivo workflow referente ao processo do login respectivamente:

Diagrama

Descrição gerada automaticamente

**Figura 4 – Diagrama UML de Sequência do Use Case Depositar do Caixa Eletrônico**

**3**

**III.2.4 – A Funcionalidade Saldo da CLASSE CaixaEletronico:**

A partir das **especificações de requisitos da TAREFA da Semana três** providas pela **Plataforma Cousera/ITA** para o desenvolvimento Software de Caixa Eletrônico, foi possível elaborar o *Diagrama UML de Sequência,* que mostra a colaboração das classes na *Funcionalidade SALDO do Caixa Eletrônico*, e toda a sequência de operações , incluindo a Classe de Teste da Classe Caixa Eletrônico e respectivo workflow referente ao processo do login respectivamente:

Diagrama, Linha do tempo

Descrição gerada automaticamente

**Figura 5 – Diagrama UML de Sequência do Use Case Saldo do Caixa Eletrônico**

**3**

**III.3 – DIAGRAMA UML de CLASSES do SOFTWARE de CAIXA ELETRÔNICO:**

A partir das **especificações de requisitos da TAREFA da Semana três** providas pela **Plataforma Cousera/ITA** para o desenvolvimento do Software de Caixa Eletrônico, foi possível elaborar o *Diagrama UML de Classes,* que mostra o *Design da Arquitetura das Classes e colaborações da Aplicação Caixa Eletrônico*, respectivamente:

Diagrama

Descrição gerada automaticamente

**Figura 7 – Diagrama de Classes UML do Software de Caixa Eletrônico**

**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* *devo 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:

Diagrama

Descrição gerada automaticamente

**Figura 8 –** *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.

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 a mesma 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.

**V – INFRAESTRUTURA UTILIZADA NO 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:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Classes e Interfaces do Software de Caixa Eletrônico**

**Classes de Testes com o Suite de casos de testes unitários do Software de Caixa Eletrônico**

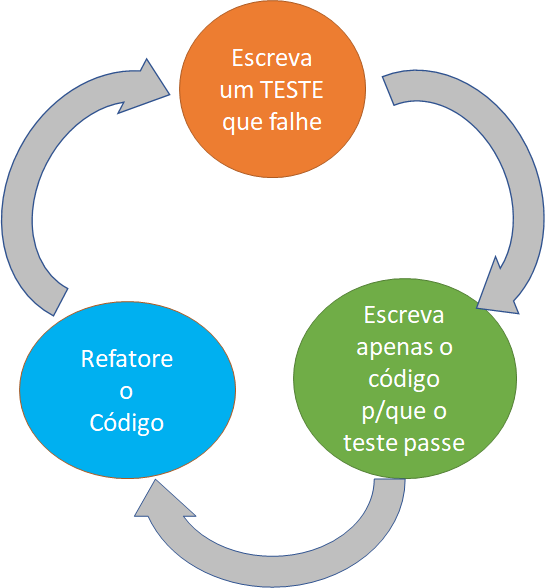
**Figura 9 – Estrutura de Folders do Projeto JAVA**

**VI – 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 10 – O CICLO do TDD utilizado no desenvolvimento**

**VII – PLANEJAMENTO dos CASOS de TESTES UNITÁRIOS do SOFTWARE de CAIXA ELETRÔNICO:**

**VIII.1 – Introdução**

Com a finalidade de atender a prática do TDD, houve necessidade de ter um planejamento mínimo dos casos de teste da aplicação CaixaEletronico, que correspondem a previsão dos casos com dados amostrais que atendem as interfaces de cada uma das funcionalidades implementadas do software de Caixa Eletrônico, além de prever as situações anormais que poderiam comprometer o funcionamento da aplicação e impedir que a mesma crash durante o seu funcionamento e durante o uso pelos usuários finais.

**VIII.2 – Restrição na Execução dos Casos de Testes – Ordem Específica dos Cases de Testes p/JUnit 4**

Neste caso específico dos testes desta aplicação CaixaEletrônico que usa a Classe ContaCorrente, onde precisamos ter um ambiente previsível e ter a criação das contas correntes antes de tudo, para então, podermos realizar saques, depósitos e consultas.

Se você observar, o Junit não executa os testes na ordem com que os métodos são criados dentro das Classes de Testes. Olhe atentamente e verá que, os testes são executados aleatoriamente – isto é, o JUnit por “default” não segue nenhuma ordem especifica para executar os casos de testes, portanto, não sendo previsível que ordem adotará e com isso podem comprometer o fluxo da sequência dos testes e gerar erros inesperados nos códigos dos métodos de nossas classes.

Logo, precisamos controlar o **JUnit Test Execution Order***.*  Há diferentes maneiras e modos disponíveis para definir a ordem de execução dos casos de testes na plataformas de testes unitários JUnit versões 4 e 5 – em cada versão há diferenças das anotações e usar e como fazer.

A escolha que adotei foi que os métodos de testes fossem executados na ordem lexicográfica, isto é, pela ordem dos nomes dos métodos de testes. Isso foi feito por meio do uso da anotação MethodSorters.NAME\_ASCENDING.

Porém, com o uso deste recurso, tem um efeito que é: p*recisamos ter o nome dos testes escritos de maneira que seja possível seguir a ordem que desejada*, para *não comprometer* o planejamento dos destes de logins, saques, depósitos e consultas de saldos.

Como condição necessária, precisei inserir os seguintes imports e a seguinte anotação precedendo o nome das classes de testes, respectivamente:

***import org.junit.FixMethodOrder;***

***import org.junit.runners.MethodSorters;***

***@FixMethodOrder(MethodSorters.NAME\_ASCENDING)***

**VII.3 –Testes do Método Persistir () da Classe MockServicoRemoto**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Teste #** | **Cenario do**  **Teste** | **Número da**  **Conta** | **Saldo da**  **Conta** | **Senha da**  **Conta** |
| 1 | testa001recuperarPrimeiraContaCorrente | 124578 | 200.0 | @CC |
| 2 | testa002recuperarSegundaContaCorrente | 121212 | 50.0 | senha1 |
| 3 | testa003recuperarTerceiraContaCorrente | 9998 | 100.0 | XYZabc |
| 4 | testa004recuperarQuartaContaCorrente | 646464 | -450.0 | senha66 |
| 5 | testa005recuperarQuintaContaCorrente | 616161 | 90.0 | senha13 |
| 6 | testa006recuperarSextaContaCorrente | 414141 | 300.0 | senha21 |
| 7 | testa007recuperarSetimaContaCorrente | 232323 | 0.0 | senha33 |
| 8 | testa008recuperaroitavaContaCorrente | 515151 | 150.0 | senha99 |

**VII.2 – Testes do Método Logar () da Classe CaixaEletrônico**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Teste #** | **Cenario do**  **Teste** | **Número da**  **Conta** | **Saldo da**  **Conta** | **Senha da**  **Conta** |
| 9 | testa01PrimeiroLogin | 22222 | 200.0f | xyz |
| 10 | testa02SegundoLogin | 1234 | 100.0f | senha1 |
| 11 | testa03TerceiroLogin | 999999 | 200.0 | Senha99 |
| 12 | Testa04QuartoLogin | 777777 | 350.0 | Senha98 |
| 13 | Testa05QuintoLogin | 666666 | 190.0 | Senha97 |
| 14 | Testa06SextoLogin | 888888 | -250.0 | Senha96 |
| 15 | Testa07SetimoLogin | 555555 | 0.0 | Senha95 |

**VII.3 – Testes do Método sacar() da Classe CaixaEletrônico**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Caso de Teste #** | **Cenario do**  **Teste** | **Número da**  **Conta** | **Saldo da**  **Conta** | **Valor a**  **Sacar** | **Valor**  **Novo Saldo** | **Senha**  **da Conta** |
| 16 | testa08saqueComSaldoSuficienteTeste01 | 999999 | 200.0 | 200.0 | 0.0 | Senha99 |
| 17 | testa09saqueComSaldoSuficienteTeste02 | 777777 | 350.0 | 100,0 | 250.0 | Senha98 |
| 18 | testa10saqueSemSaldoSuficienteTeste03 | 666666 | 190.0 | 40.0 | 150.0 | Senha97 |
| 19 | testa11saqueComSaldoNegativoTeste01 | 888888 | -250.0 | 140,0 | -250.0 | Senha96 |
| 20 | testa12saqueComSaldoZeradoTeste01 | 555555 | 0.0 | 80.0 | 0.0 | Senha95 |

**VII.3 –Testes do Método depositar() da Classe CaixaEletrônico**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Caso de Teste #** | **Cenario do**  **Teste** | **Número da**  **Conta** | **Saldo da**  **Conta** | **Valor a**  **Depositar** | **Valor**  **Novo Saldo** | **Senha**  **da Conta** |
| 21 | testa13depositoContaSaldoMaiorPositivoTeste01 | 1234 | 100.0 | 500,0 | 600,0 | senha1 |
| 22 | testa14depositoContaSaldoMaiorPositivoTeste02 | 777777 | 250.0 | 50,0 | 300.0 | Senha98 |
| 23 | testa15depositoContaSaldoMaiorPositivoTeste03 | 666666 | 150.0 | 40.0 | 190.0 | Senha97 |
| 24 | testa16depositoContaComSaldoNegativoTeste01 | 888888 | -250.0 | 140,0 | -110.0 | Senha96 |
| 25 | testa17depositoComSaldoZeradoTeste01 | 555555 | 0.0 | 80.0 | 80.0 | Senha95 |
| 26 | testa18depositoComSaldoZeradoTeste02 | 999999 | 0.0 | 100.0 | 100.0 | Senha99 |
| 27 | testa19depositoEmContaQueTeveDepositoTeste01 | 555555 | 80.0 | 80.0 | 180.0 | Senha95 |
| 28 | Testa20depositoEmContaQueTeveDepositoTeste02 | 666666 | 190.0 | 40.0 | 230.0 | Senha97 |

**VIII – 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

**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;  **import** org.junit.FixMethodOrder;  **import** org.junit.runners.MethodSorters;  **@FixMethodOrder(MethodSorters**.***NAME\_ASCENDING***)  **public** **class** MockServicoRemotoTest {  MockServicoRemoto \_mock = **new** MockServicoRemoto();  **@Test**  **public** **void** testa001recuperarPrimeiraContaCorrente( ) {  \_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** testa002recuperarSegundaContaCorrente( ) {  \_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** testa003recuperarTerceiraContaCorrente( ) {  \_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** testa004recuperarQuartaContaCorrente( ) {  \_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** testa005recuperarQuintaContaCorrente( ) {  \_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** testa006recuperarSextaContaCorrente( ) {  \_mock.persistirConta(**new** ContaCorrente ("515151", 150.0f, "senha99"));  ContaCorrente \_cc = \_mock.recuperarConta("515151");  *assertEquals* ("515151", \_cc.getNumeroConta());  *assertEquals* (150.0f, \_cc.getSaldo());  *assertEquals* ("senha99", \_cc.getSenha());  }  **@Test**  **public** **void** testa007recuperarSetimaContaCorrente( ) {  \_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** testa008recuperarOitavaContaCorrente( ) {  \_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());  }  } |

**B.2 – Classe CaixaEletronicoTest**

|  |
| --- |
| **package** cxeletronico;  **import** **static** org.junit.Assert.\*;  **import** org.junit.Before;  **import** org.junit.Test;  **import** org.junit.FixMethodOrder;  **import** org.junit.runners.MethodSorters;  @FixMethodOrder(MethodSorters.***NAME\_ASCENDING***)  **public** **class** CaixaEletronicoTest {  **private** MockHardware \_mockHD = **new** MockHardware();  **private** MockServicoRemoto \_mockSR = **new** MockServicoRemoto();  **private** CaixaEletronico \_cxe = **new** CaixaEletronico ();  @Before  **public** **void** init() {  \_cxe.adicionarHardware(\_mockHD);  \_cxe.adicionarServicoRemoto(\_mockSR);  }  @Test  **public** **void** testa01PrimeiroLogin() {  System.***out***.println("\n"+" CaixaEletronicoTest t009 testa01PrimeiroLogin( ) ");  \_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.logar());  }  @Test  **public** **void** testa02SegundoLogin() {  System.***out***.println("\n"+" CaixaEletronicoTest t010 testa02SegundoLogin( ) ");  \_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.logar());  }  @Test  **public** **void** testa03TerceiroLogin() {  System.***out***.println("\n"+" CaixaEletronicoTest t011 testa03TerceiroLogin( ) ");  \_mockSR.persistirConta(**new** ContaCorrente ("999999", 200.0f, "Senha99"));  ContaCorrente \_cc = \_mockSR.recuperarConta("999999");  *assertEquals* ("999999", \_cc.getNumeroConta());  *assertEquals* (200.0f, \_cc.getSaldo());  *assertEquals* ("Senha99", \_cc.getSenha());  *assertEquals* ( **true**, \_cxe.logar());  }  @Test  **public** **void** testa04QuartoLogin() {  System.***out***.println("\n"+" CaixaEletronicoTest t012 testa04QuartoLogin( ) ");  \_mockSR.persistirConta(**new** ContaCorrente ("777777", 350.0f, "Senha98"));  ContaCorrente \_cc = \_mockSR.recuperarConta("777777");  *assertEquals* ("777777", \_cc.getNumeroConta());  *assertEquals* (350.0f, \_cc.getSaldo());  *assertEquals* ("Senha98", \_cc.getSenha());  *assertEquals* ( **true**, \_cxe.logar());  }  @Test  **public** **void** testa05QuintoLogin() {  System.***out***.println("\n"+" CaixaEletronicoTest t013 testa05QuintoLogin( ) ");  \_mockSR.persistirConta(**new** ContaCorrente ("666666", 190.0f, "Senha97"));  ContaCorrente \_cc = \_mockSR.recuperarConta("666666");  *assertEquals* ("666666", \_cc.getNumeroConta());  *assertEquals* (190.0f, \_cc.getSaldo());  *assertEquals* ("Senha97", \_cc.getSenha());  *assertEquals* ( **true**, \_cxe.logar());  }  @Test  **public** **void** testa06SextoLogin() {  System.***out***.println("\n"+" CaixaEletronicoTest t014 testa06SextoLogin( ) ");  \_mockSR.persistirConta(**new** ContaCorrente ("888888", -250.0f, "Senha96"));  ContaCorrente \_cc = \_mockSR.recuperarConta("888888");  *assertEquals* ("888888", \_cc.getNumeroConta());  *assertEquals* (-250.0f, \_cc.getSaldo());  *assertEquals* ("Senha96", \_cc.getSenha());  *assertEquals* ( **true**, \_cxe.logar());  }  @Test  **public** **void** testa07SetimoLogin() {  System.***out***.println("\n"+" CaixaEletronicoTest t015 testa07SetimoLogin( ) ");  \_mockSR.persistirConta(**new** ContaCorrente ("555555", 0.0f, "Senha95"));  ContaCorrente \_cc = \_mockSR.recuperarConta("555555");  *assertEquals* ("555555", \_cc.getNumeroConta());  *assertEquals* (0.0f, \_cc.getSaldo());  *assertEquals* ("Senha95", \_cc.getSenha());  *assertEquals* ( **true**, \_cxe.logar());  }  @Test  **public** **void** testa08saqueComSaldoSuficienteTeste01() {  System.***out***.println("\n"+" CaixaEletronicoTest t016 testa08saqueComSaldoSuficienteTeste01( ) ");  *assertEquals* ( **true**, \_cxe.sacar("999999", 200.0f));  ContaCorrente \_cc = \_mockSR.recuperarConta("999999");  *assertEquals* ("999999", \_cc.getNumeroConta());  *assertEquals* (0.0f, \_cc.getSaldo());  *assertEquals* ("Senha99", \_cc.getSenha());  }  @Test  **public** **void** testa09saqueComSaldoSuficienteTeste02() {  System.***out***.println("\n"+" CaixaEletronicoTest t017 testa09saqueComSaldoSuficienteTeste02( ) ");  *assertEquals* ( **true**, \_cxe.sacar("777777", 100.0f));  ContaCorrente \_cc = \_mockSR.recuperarConta("777777");  *assertEquals* ("777777", \_cc.getNumeroConta());  *assertEquals* (250.0f, \_cc.getSaldo());  *assertEquals* ("Senha98", \_cc.getSenha());  }  @Test  **public** **void** testa10saqueComSaldoSuficienteTeste03() {  System.***out***.println("\n"+" CaixaEletronicoTest t018 testa10saqueComSaldoSuficienteTeste03( ) ");  *assertEquals* ( **true**, \_cxe.sacar("666666", 40.0f));  ContaCorrente \_cc = \_mockSR.recuperarConta("666666");  *assertEquals* ("666666", \_cc.getNumeroConta());  *assertEquals* (150.0f, \_cc.getSaldo());  *assertEquals* ("Senha97", \_cc.getSenha());  }  @Test  **public** **void** testa11saqueComSaldoNegativoTeste01() {  System.***out***.println("\n"+" CaixaEletronicoTest t019 testa11saqueComSaldoNegativoTeste01( ) ");  *assertEquals* ( **false**, \_cxe.sacar("888888", 140.0f));  ContaCorrente \_cc = \_mockSR.recuperarConta("888888");  *assertEquals* ("888888", \_cc.getNumeroConta());  *assertEquals* (-250.0f, \_cc.getSaldo());  *assertEquals* ("Senha96", \_cc.getSenha());  }  @Test  **public** **void** testa12saqueComSaldoZeradoTeste01() {  System.***out***.println("\n"+" CaixaEletronicoTest t020 testa12saqueComSaldoZeradoTeste01( ) ");  *assertEquals* ( **false**, \_cxe.sacar("555555", 80.0f));  ContaCorrente \_cc = \_mockSR.recuperarConta("555555");  *assertEquals* ("555555", \_cc.getNumeroConta());  *assertEquals* (0.0f, \_cc.getSaldo());  *assertEquals* ("Senha95", \_cc.getSenha());  }  @Test  **public** **void** testa13depositoContaSaldoMaiorPositivoTeste01() {  System.***out***.println("\n"+" CaixaEletronicoTest t021 testa13depositoContaSaldoMaiorPositivoTeste01( ) ");  *assertEquals* ( **true**, \_cxe.depositar("1234", 500.0f));  ContaCorrente \_cc = \_mockSR.recuperarConta("1234");  *assertEquals* ("1234", \_cc.getNumeroConta());  *assertEquals* (600.0f, \_cc.getSaldo());  *assertEquals* ("senha1", \_cc.getSenha());  }  @Test  **public** **void** testa14depositoContaSaldoMaiorPositivoTeste02() {  System.***out***.println("\n"+" CaixaEletronicoTest t022 testa14depositoContaSaldoMaiorPositivoTeste02( ) ");  *assertEquals* ( **true**, \_cxe.depositar("777777", 50.0f));  ContaCorrente \_cc = \_mockSR.recuperarConta("777777");  *assertEquals* ("777777", \_cc.getNumeroConta());  *assertEquals* (300.0f, \_cc.getSaldo());  *assertEquals* ("Senha98", \_cc.getSenha());  }  @Test  **public** **void** testa15depositoContaSaldoMaiorPositivoTeste03() {  System.***out***.println("\n"+" CaixaEletronicoTest t023 testa15depositoContaSaldoMaiorPositivoTeste03( ) ");  *assertEquals* ( **true**, \_cxe.depositar("666666", 40.0f));  ContaCorrente \_cc = \_mockSR.recuperarConta("666666");  *assertEquals* ("666666", \_cc.getNumeroConta());  *assertEquals* (190.0f, \_cc.getSaldo());  *assertEquals* ("Senha97", \_cc.getSenha());  }  @Test  **public** **void** testa16depositoContaComSaldoNegativoTeste01() {  System.***out***.println("\n"+" CaixaEletronicoTest t024 testa16depositoContaComSaldoNegativoTeste01( ) ");  *assertEquals* ( **true**, \_cxe.depositar("888888", 140.0f));  ContaCorrente \_cc = \_mockSR.recuperarConta("888888");  *assertEquals* ("888888", \_cc.getNumeroConta());  *assertEquals* (-110.0f, \_cc.getSaldo());  *assertEquals* ("Senha96", \_cc.getSenha());  }  @Test  **public** **void** testa17depositoComSaldoZeradoTeste01() {  System.***out***.println("\n"+" CaixaEletronicoTest t025 testa17depositoComSaldoZeradoTeste01( ) ");  *assertEquals* ( **true**, \_cxe.depositar("555555", 80.0f));  ContaCorrente \_cc = \_mockSR.recuperarConta("555555");  *assertEquals* ("555555", \_cc.getNumeroConta());  *assertEquals* (80.0f, \_cc.getSaldo());  *assertEquals* ("Senha95", \_cc.getSenha());  }  @Test  **public** **void** testa18depositoComSaldoZeradoTeste02() {  System.***out***.println("\n"+" CaixaEletronicoTest t026 testa18depositoComSaldoZeradoTeste02( ) ");  *assertEquals* ( **true**, \_cxe.depositar("999999", 100.0f));  ContaCorrente \_cc = \_mockSR.recuperarConta("999999");  *assertEquals* ("999999", \_cc.getNumeroConta());  *assertEquals* (100.0f, \_cc.getSaldo());  *assertEquals* ("Senha99", \_cc.getSenha());  }  @Test  **public** **void** testa19depositoEmContaQueTeveDepositoTeste01() {  System.***out***.println("\n"+" CaixaEletronicoTest t027 testa19depositoEmContaQueTeveDepositoTeste01( ) ");  *assertEquals* ( **true**, \_cxe.depositar("555555", 80.0f));  ContaCorrente \_cc = \_mockSR.recuperarConta("555555");  *assertEquals* ("555555", \_cc.getNumeroConta());  *assertEquals* (160.0f, \_cc.getSaldo());  *assertEquals* ("Senha95", \_cc.getSenha());  }  @Test  **public** **void** Testa20depositoEmContaQueTeveDepositoTeste02() {  System.***out***.println("\n"+" CaixaEletronicoTest t028 Testa20depositoEmContaQueTeveDepositoTeste02( ) ");  *assertEquals* ( **true**, \_cxe.depositar("666666", 40.0f));  ContaCorrente \_cc = \_mockSR.recuperarConta("666666");  *assertEquals* ("666666", \_cc.getNumeroConta());  *assertEquals* (190.0f, \_cc.getSaldo());  *assertEquals* ("Senha97", \_cc.getSenha());  }      } |

**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);  } |

**C.2 – Interface ServicoRemoto**

|  |
| --- |
| **package** cxeletronico;  **public** **interface** IServicoRemoto {  **public** ContaCorrente recuperarConta(String numeroConta);  **public** **void** persistirConta(ContaCorrente cc);    } |

**C.3 – Interface IHardwareComplemento**

|  |
| --- |
| **package** cxeletronico;  **public** **interface** IHardwareComplemento **extends** IHardware {    **public** **void** exibirMsgAoUsuarioCaixaEletronico(String msg);  **public** String solicitarSenhaDoUsuario (String msg);  **public** String solicitarValorASacar (String msg);  **public** String solicitarValorADepositar(String msg);    } |

**C.4 – 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;  }    **public** **void** salvaSaldoAposSaqueOuDeposito(**float** novoSaldo) {  \_saldoConta = novoSaldo;  }    **public** **void** abaterValorSaqueDoSaldo(**float** valorDoSaque) {  \_saldoConta -= valorDoSaque;  }    **public** **void** adicionarValorDepositoAoSaldo(**float** valorDoDeposito) {  \_saldoConta += valorDoDeposito;  }    } |

**C.5 – Classe CaixaEletronico**

|  |
| --- |
| package cxeletronico;  import java.util.ArrayList;  import java.util.List;  public class CaixaEletronico {  private List<IHardwareComplementar> \_hardware = new ArrayList();  private List<IServicoRemoto> \_servRemoto = new ArrayList();  private IHardwareComplementar \_h;  private IServicoRemoto \_s;  private ContaCorrente \_cc;    public Boolean logar() {  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.pegarNumeroDaConta("SemErro");  \_cc = \_s.recuperarConta(\_numeroConta);  \_senhaUsuario = \_h.solicitarSenhaDoUsuario("SemErro");  String \_senhaGravada = \_cc.getSenha();  if ( !\_senhaUsuario.contains(\_senhaGravada) ) {  \_h.exibirMsgAoUsuarioCaixaEletronico(\_LOGIN\_FALHOU);  return false;  }  } catch (Exception e) {  \_h.exibirMsgAoUsuarioCaixaEletronico(\_LOGIN\_FALHOU);  return false;  }  \_h.exibirMsgAoUsuarioCaixaEletronico(\_LOGIN\_OK);  return true;  }  public void adicionarHardware(IHardwareComplementar hardwareCXE) {  \_hardware.clear();  \_hardware.add(hardwareCXE);  // System.out.println("MOCK HARDWARE ADICIONADO");  }  public void adicionarServicoRemoto(IServicoRemoto servRemoto) {  \_servRemoto.clear();  \_servRemoto.add(servRemoto);  // System.out.println("MOCK SERVIDOR REMOTO ADICIONADO");  }    public Boolean sacar(String numConta, float valorDoSaque) {  System.out.println("Sacar de CC: " + numConta );  System.out.println(" Saque = " + valorDoSaque);  if (\_hardware.isEmpty())  throw new RuntimeException ("Erro Interno Aplicacao");  if (\_servRemoto.isEmpty())  throw new RuntimeException ("Erro Interno Aplicacao");  \_h = \_hardware.get(0);  \_s = \_servRemoto.get(0);  \_cc = \_s.recuperarConta(numConta);  float \_saldoCliente = (float) \_cc.getSaldo();  System.out.println(" Saldo Atual = " + \_saldoCliente);  try {  if (\_saldoCliente < valorDoSaque) {  \_h.exibirMsgAoUsuarioCaixaEletronico("Saldo Insuficinte");  System.out.println(" Saldo Final = " + \_cc.getSaldo());  return false;  }  \_cc.abaterValorSaqueDoSaldo(valorDoSaque);  System.out.println(" Saldo Final = " + \_cc.getSaldo());  \_s.persistirConta (\_cc);  \_h.exibirMsgAoUsuarioCaixaEletronico("Retire seu Dinheiro");    } catch (Exception e) {  \_h.exibirMsgAoUsuarioCaixaEletronico("Problema no Hardware");  return false;  }  return true;  }  public Boolean depositar(String numConta, float valorADepositar) {  System.out.println("Deposita na CC: " + numConta );  System.out.println("Deposito de = " + valorADepositar);  if (\_hardware.isEmpty())  throw new RuntimeException ("Erro Interno Aplicacao");  if (\_servRemoto.isEmpty())  throw new RuntimeException ("Erro Interno Aplicacao");  \_h = \_hardware.get(0);  \_s = \_servRemoto.get(0);  try {  \_cc = \_s.recuperarConta(numConta);  System.out.println(" Saldo Atual = " + \_cc.getSaldo());  \_cc.adicionarValorDepositoAoSaldo( valorADepositar );  \_h.exibirMsgAoUsuarioCaixaEletronico("Insira o Envelope");  \_s.persistirConta (\_cc);  System.out.println(" Saldo Final = " + \_cc.getSaldo());  \_h.exibirMsgAoUsuarioCaixaEletronico("Deposito Recebido com Sucesso");  } catch (Exception e) {  \_h.exibirMsgAoUsuarioCaixaEletronico("Problema no Hardware");  return false;  }  return true;  }    } |

**C.6 – 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) {  System.out.println("Conta a Recuperar= (" + numeroConta + ")");  System.out.println("Ocorrencias da ContaCorrente= (" + \_contas.size() + ")");  for (int i = 0; i < \_contas.size(); i++) {  System.out.println(" SERVREMOTO INDICE DATABASE= (" + i );  String \_numConta = \_contas.get(i).getNumeroConta();  if ( \_numConta.contains(numeroConta) ) {  System.out.println(" SERVREMOTO NumConta= (" + \_contas.get(i).getNumeroConta() +  ") Saldo= (" + \_contas.get(i).getSaldo() +  ") Senha= (" + \_contas.get(i).getSenha() + ")");  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) {  ContaCorrente \_cctemp = cc;  String \_numConta = cc.getNumeroConta();  float \_novoSaldo = (float) cc.getSaldo();  int \_contaNaoExiste = 1;  for (int i = 0; i < \_contas.size(); i++) {  String \_nConta = \_contas.get(i).getNumeroConta();  if ( \_numConta.contains(\_nConta) ) {  \_contas.get(i).salvaSaldoAposSaqueOuDeposito(\_novoSaldo);  \_contaNaoExiste = 0;  }  }  if (\_contaNaoExiste == 1)  \_contas.add(cc);  }  public String devolverNumConta(int numRegistro) {  if ( \_contas.isEmpty())  throw new RuntimeException ("Erro - Database Vazio!!!");  if (\_contas.size() < numRegistro)  throw new RuntimeException ("Erro - Database Vazio!!!");  return (String)\_contas.get(numRegistro).getNumeroConta();  }    public String devolverSenhaConta(int numRegistro) {  if ( \_contas.isEmpty())  throw new RuntimeException ("Erro - Database Vazio!!!");  if (\_contas.size() < numRegistro)  throw new RuntimeException ("Erro - Database Vazio!!!");  return (String)\_contas.get(numRegistro).getSenha();  }    } |
|  |

**C.7 – Classe MockHardware**

|  |
| --- |
| **package** cxeletronico;  **public** **class** MockHardware **implements** IHardwareComplementar {  **private** **static** **int** *\_indice*=-1;  **private** **int** \_INDEX\_MAXIMO\_CONTAS = 14;  @Override  **public** String pegarNumeroDaConta(String msg) {  **if** (msg == **null**)  **throw** **new** RuntimeException ("Problema de Hardware");  **if** (*\_indice* > \_INDEX\_MAXIMO\_CONTAS || *\_indice* < 0)  *\_indice* = 0; **else** *\_indice*++;  System.***out***.println(" HARDWARE indice: " + *\_indice* );  MockServicoRemoto \_mockSR = **new** MockServicoRemoto();  **return** (String) \_mockSR.devolverNumConta(*\_indice*);  }  @Override  **public** **void** entregarDinheiro(String msg) {  exibirMsgAoUsuarioCaixaEletronico (msg);  };  @Override  **public** **void** lerEnvelope(String msg) {  **if** (msg == **null**)  **throw** **new** RuntimeException ("Problema de Hardware");  exibirMsgAoUsuarioCaixaEletronico (msg);  }  @Override  **public** **void** exibirMsgAoUsuarioCaixaEletronico(String msg) {  **if** (msg == **null**)  **throw** **new** RuntimeException ("Problema de Hardware");  System.***out***.println("msg: " + msg);  }  @Override  **public** String solicitarSenhaDoUsuario (String msg) {  **if** (msg == **null**)  **throw** **new** RuntimeException ("Problema de Hardware");  // System.out.println(" HARDWARE indice: " + \_indice );  MockServicoRemoto \_mockSR = **new** MockServicoRemoto();  **return** (String) \_mockSR.devolverSenhaConta(*\_indice*);  }  @Override  **public** String solicitarValorASacar(String msg) {  // **TODO** Auto-generated method stub  **return** **null**;  }  @Override  **public** String solicitarValorADepositar(String msg) {  // **TODO** Auto-generated method stub  **return** **null**;  }  } |

**IX – 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.

**X – 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.